CipherLab BASIC Compiler User's Manual

Ver. 2.02

Copyright © 2000 Syntech Information Co., Ltd.

SYNTECH INFORMATION CO., LTD.

Head Office: 8F, No.210, Ta-Tung Rd., Sec.3, Hsi-Chih, Taipei Hsien, Taiwan
Tel: +886-2-2643-8866 Fax: +886-2-2643-8800
e-mail: support@cipherlab.com.tw http://www.cipherlab.com.tw

TABLE OF CONTENTS

Preface iv
1 Development Environment 1
1.1 Disk Contents 1
1.2 System Requirements 2
1.3 Development Flow 3
1.3.1 Download Run-time Engine 3
1.3.2 Edit and Compile the BASIC Program 4
1.3.3 Download the BASIC Object Files 4
1.4 BASIC Run-time Engine 5
2 Using CipherLab BASIC Compiler 8
2.1 File Menu 8
2.2 Edit Menu 8
2.3 Config Menu 9
2.4 Compile Menu 10
2.5 Help Menu 11
3 Basics of CipherLab BASIC Language 12
3.1 Constants 12
3.1.1 String 12
3.1.2 Numeric 12
3.2 Variables 13
3.2.1 Variable Names and Declaration Characters 13
3.2.2 Array Variables 13
3.3 Expression and Operators 14
3.3.1 Assignment Operator 14
3.3.2 Arithmetic Operator 14
3.3.3 Relational Operator 14
3.3.4 Logical Operator 15
3.4 Operator Precedence 16
3.5 Labels 17
3.6 Subroutines 18
3.7 Programming Style 19
4 BASIC Commands 20
4.1 General Commands 21
4.2 Commands for Decision Structures 24
4.3 Commands for Looping Structures 27
4.4 Commands for String Processing 29
4.4.1 Combining Strings 29
4.4.2 Comparing Strings 29
4.4.3 Getting the Length of a String 29
4.4.4 Searching for Strings 30
4.4.5 Retrieving Part of Strings 30
4.4.6 Converting for Strings 32
4.4.7 Creating Strings of Repeating Characters 34
4.5 Commands for Event Trapping 35
4.5.1 Event Triggers 35
4.5.2 Lock and Unlock 44
4.6 System Commands 45
4.7 Reader Commands 47
4.8 Keyboard Wedge Commands 53
4.8.1 Definition of the *WedgeSetting* array 53
4.8.2 KBD / Terminal Type 53
4.8.3 Capital Lock Auto-Detection 54
4.8.4 Capital Lock Status Setting 54
4.8.5 Alphabets Case 54

4.8.6 Digits Position 54
4.8.7 Shift / Capital Lock Keyboard 55
4.8.8 Digit Transmission 55
4.8.9 Inter-Character Delay 55
4.8.10 Composition of Output String 55
4.8.11 Special Note on DEC VT220/320/420 (for 201 only) 56
4.9 Buzzer 58
4.10 Calendar and Timer Commands 60
4.11 Digital I/O Commands 63
4.12 LED Command 64
4.13 Keyboard Commands 66
4.14 LCD Commands 70
4.14.1 Graphic Display 70
4.14.2 Special Font Files 70
4.15 Battery Commands 75
4.16 Communication Ports 76
4.16.1 RS-232 Communications 76
4.16.2 RS-485 Communications 80
4.17 File Manipulation 84
4.17.1 DAT Files 84
4.17.2 DBF Files and IDX Files 89
4.17.3 Error Code 95
4.18 Memory 96
4.19 Debug Commands 97
4.20 Reserved Host Commands 99
5 Sample Programs 102
5.1 201TEST_1.BAS 102
5.2 201TEST_2.BAS 104
5.3 201TEST_3.BAS 106
5.4 510AC_100.BAS 108
5.5 711AM_100.BAS 114
Appendix A: Version History 121
Appendix B: Flash Writer 123
Appendix C: Run-Time Error Table 124
Appendix D: Barcode Setting 125
D.1. Symbology Parameters 125
D.1.1. Code 39 125
D.1.2. Italy / French Phamacode 125
D.1.3. Industrial / Interleave / Matrix 2 of 5 125
D.1.4. Codabar 126
D.1.5. MSI 126
D.1.6. Plessey 126
D.1.7. UPCE 126
D.1.8. EAN8 127
D.1.9. EAN13 & UPCA 127
D.2. Scanner Parameters 127
D.2.1. Scan Mode 127
D.2.2. Read Redundancy 128
D.2.3. Time-Out 128
D.2.4. Negative Barcode 128
Appendix E: Pseudo Keyboard 129
Appendix F: Debug Messages 132
Index 138

Preface

CipherLab BASIC Compiler provides users with a complete programming environment to develop application programs for the CipherLab terminals in the BASIC language. The Windows-based Basic Compiler comes with a menu-driven interface to simplify software development and code modifications. Many system configurations, such as the master set-up card ID and barcode settings can be set up in the menus. Using this powerful programming tool, users can develop an application for their own needs, without lengthy coding.

This manual is intended to provide detailed information about using the BASIC Compiler. It is divided into five chapters. The outline for each chapter follows:

- Chapter 1, “Development Environment”, gives a concise introduction about the Basic Compiler, the development flow for applications, and the BASIC Compiler Run-time Engines.
- Chapter 2, “Using CipherLab BASIC Compiler”, gives a tour of the programming environment of the BASIC Compiler.
- Chapter 3, “Basics of CipherLab BASIC Language”, discusses the specific characteristics of the CipherLab BASIC Language.
- Chapter 4, “BASIC Commands”, discusses all the supported BASIC functions and statements. More than 170 BASIC functions and statements are categorized according to their functions, and discussed in details.
- Chapter 5, “Sample Programs”, contains several sample programs for readers' reference. Users can learn the usage of the BASIC functions and statements from these five sample programs.

CipherLab BASIC Compiler has been modified and improved since it first released in November 1997. Users can refer to Appendix A for version history.

1 Development Environment

1.1 Disk Contents

The CipherLab BASIC Compiler Kit contains two directories, namely, BC and DOWNLOAD. The purposes/contents of each directory are listed below.

(1) BC: This directory contains the BASIC Compiler and the BASIC programs.

- Bc.exe: the BASIC Compiler.
- Bc.hlp: the on-line help file for the BASIC Compiler.
- Bc.cnt: the content file of the BASIC on-line help.
- Synload.exe: the download program to download the Basic object files, ***.syn*** and ***.ini***, to the CipherLab terminals.
- BASIC source files (.bas) of the sample programs:

  201TEST_1.bas, 201TEST_2.bas, 201TEST_3.bas,
  201TEST_4.bas, 510AC_100.bas, 711AM_100.bas
- System initialization files (.ini) of the sample programs:

  201TEST_1.ini, 201TEST_2.ini, 201TEST_3.ini,
  201TEST_4.ini, 510AC_100.ini, 711AM_100.ini
- BASIC object files (.syn) of the sample programs:

  201TEST_1.syn, 201TEST_2.syn, 201TEST_3.syn,
  201TEST_4.syn, 510AC_100.syn, 711AM_100.syn

(2) DOWNLOAD: This directory contains a download program, the BASIC Compiler Run-time Engines and the special font files.

- Download.exe: the download program to download the Motorola S format object file (.shx) to the CipherLab terminals via RS-232 or standard IrDA port.
- Irload.exe: the download program to download the Motorola S format object file (.shx) to the CipherLab terminals via Serial IR Transceiver.
- BASIC Run-time Engines for different terminals:

  BC201.shx, BC510.shx, BC520.shx // English version
  BC520-SC.shx, // Simplified Chinese version
  BC520-TC.shx, // Traditional Chinese version
  BC711.shx, BC720.shx, // English version
  BC711-JP.shx, BC720-JP.shx, // Japanese version
  BC711-SC.shx, BC720-SC.shx, // Simplified Chinese, font size: 16x16
  BC711-SD.shx, BC720-SD.shx, // Simplified Chinese, font size: 12x12
  BC711-TC.shx, BC720-TC.shx, // Traditional Chinese version
- Font Files for 520:

  FN520-sc.shx, // Simplified Chinese, font size: 16x16
  FN520-tc.shx, // Traditional Chinese, font size: 16x16
- Font Files for 711/720:

  Font-jp.shx, // Kanji font
  Font-sc.shx, // Simplified Chinese, font size: 16x16
  Font-sd.shx, // Simplified Chinese, font size: 12x12
  Font-tc.shx, // Traditional Chinese, font size: 16x16

To set up the BASIC programming environment on your PC, simply copy these two mentioned directories from the CD-ROM to the local hard disk.

1.2 System Requirements

Before you install CipherLab BASIC Compiler, please make sure that your PC meets the minimum requirements. The minimum requirements are listed below:

Items	Requirements
CPU	Pentium 75MHz
Operating System	Windows 95
Minimum RAM	16 MB
Minimum Hard Disk Space	20 MB

Please Note:

Any terminal being programmed will need to have a minimum 128KB Ram.

1.3 Development Flow

Developing a BASIC program for the CipherLab terminal is as simple as counting 1-2-3. There are three steps:

- Step 1 – download the BASIC Run-time Engine to the target terminal.
- Step 2 – edit and compile the BASIC program.
- Step 3 – download the BASIC object file to the target terminal.

1.3.1 Download Run-time Engine

The BASIC Run-time Engines are programs being preloaded on the CipherLab terminals to execute the BASIC object files. They must exist in the terminals before the BASIC object files are downloaded. For detailed information of the BASIC Run-time Engines, please refer to the relevant section.

To download the Run-time Engine (and/ or any other programs), the target terminal needs to be set to the "Download Mode" first to receive the new program. For fixed terminals, such as 201, 510 and 520, the program is downloaded via the RS-232 port. For the 7xx portable terminals, the program is downloaded via RS-232, Docking, IR or IrDA port. The ways to set to the "Download Mode" for different models of terminals may be somewhat different. They are discussed separately below.

(1) **201/510**

The way to set 201/510 to the "Download Mode" depends on the existing program. The COM port properties for download are defined by the existing program as well. User should refer to the existing program to find out how to download a new program.

For the 201 with ben.shx (the 201 standard program), user can scan the setup barcode to enter the "Setup Menu" and then toggle the menus to enter the "Download Mode". The new program can be downloaded through C2 or AUX 232 port, with the COM port properties as 38400 bauds, no parity, 8 data bits, and CTS/RTS handshaking.

For the 510 with 510T&A_150.shx (the 510 standard program), user can power on the terminal with the keys '3', 'SHIFT', 'ENTER' pressed at the same time to enter the "Download Mode". The new program can be downloaded through COM1 or COM3 port, with the COM port properties as 38400 bauds, no parity, 8 data bits, and no handshaking.

In case where the existing program does not provide the download function or the existing program is corrupted for any reason, the user can use the 201/510 Flash Writer to download a new program. Please refer to Appendix B for detailed instructions.

(2) **520**

Both the "Kernel Menu" and the "System Menu" provide the download function. The user can press F5, F6, F7, and F8 at power-on to enter the "Kernel Menu". In "Kernel Menu", select "1 Program Download" and then select COM port and baud rate to download the new program. Alternatively the user can press F1, F2, F3 at power-on to enter the "System Menu". In "System Menu", select "6 Version" and then "2 Flash Download" and then select baud rate to download the new program.

The "System Menu" is automatically built within the user program when the user program links 520lib.lib. In rare cases, where the existing program is corrupted this may not work. But the user can always download a program through the "Kernel Menu".

(3) **7xx**

The 7xx portable terminals have the “Kernel Menu” and the “System Menu”, that is very similar to 520. Both the “Kernel Menu” and the “System Menu” provide the download function. The user can press ‘7’ and ‘9’ at power-on to enter the “System Menu”, and then select “6 Download” to download the new program. Alternatively, the user can press ‘7’ and ‘1’ at power-on to enter the “Kernel Menu”, and select “1 Program Download” to download the new program. The Flash Writer is not needed for 7xx terminals.

Download.exe is a Windows-based program to download the Motorola S format object file (**.shx**) to the CipherLab terminals via RS-232 or IR Transceiver. After the target terminal is set to the “Download Mode” and the connection between the host PC is properly established, the user can execute this program on the host PC to download the Run-time Engine or any other ***.shx*** files to the terminal.

After the Run-time Engine is downloaded successfully, the message “Ready for BASIC Download” will be shown on the LCD of the terminal.

1.3.2 Edit and Compile the BASIC Program

The BASIC Compiler, ***bc.exe***, comes with a text editor where users can edit their BASIC programs. Please refer to the next chapter for general information of the operation.

By default, the text edited with the editor would be saved as a BASIC source (**.bas**) file. The system settings defined in the configuration menus, including “Target Machine”, “Master Card ID”, COM port settings, transaction file settings, DBF settings, barcode settings, would be saved as a system initialization (**.ini**) file with the same name when the ***.bas*** file is saved. The ***.ini*** file should be treated as part of the BASIC program, and should be included when the BASIC program is distributed.

If the BASIC program compiles with no errors, a BASIC object file (**.syn**) with the same name is generated. The ***.ini*** file and the ***.syn*** file are the two files to be downloaded to the terminal. The ***.ini*** file contains the system settings, while the ***.syn*** file contains the BASIC object code.

1.3.3 Download the BASIC Object Files

The user can use the BASIC Compiler or another download utility, such as *Synload.exe* to download a compiled BASIC program. *Synload.exe* provides only the download function of the BASIC Compiler. Using the *Synload.exe* program is provided to enable system developers to provide the end user with an easy method for reloading terminals.

Both the ***.ini*** and ***.syn*** files will be downloaded to the target terminal. Please note that if the ***.ini*** file is missing, the BASIC Compiler will download the default settings instead and this may cause errors in its execution. In contrast to the BASIC Compiler, *Synload.exe* will not process the download if the ***.ini*** file is missing, and an error message will be shown on the monitor.

After the BASIC object file is downloaded, the target terminal will reboot itself to execute the BASIC program. If any run-time error occurs, an error message will be shown on the LCD screen. Please refer to Appendix C for a list of run-time errors. If the program is not running as desired, modify the BASIC source code and download again.

1.4 BASIC Run-time Engine

The BASIC Run-time Engines work as the interpreters of the BASIC commands. The CipherLab terminals have to be preloaded with the BASIC Run-time Engines to run the BASIC programs. Each model of terminal has its own Run-time Engine to drive its specific hardware features. The Run-time Engines are named as "BCXXX.shx", where "XXX" is the model of the target terminal. For example, "BC201.shx" is the BASIC Compiler Run-time Engine for CipherLab 201 terminal.

The BASIC Run-time Engine also provides the function for the user to configure the terminal. With the Run-time Engine preloaded, the terminal can be set to the "System Mode". In the "System Mode", the user can set up the system settings, such as the system clock, and download the user program. Entering "System Mode" as well as its contents differ between the fixed and portable terminals.

(1) **201/510/520**

After the user's BASIC program is downloaded, the user can set the BASIC Compiler Run-time Engine to the "System Mode" by scanning the master setup barcode or magnetic card. The default value of the master setup barcode/magnetic card is "1234567890". The setup barcode can be printed with all possible symbologies. However, the decodability of the symbology selected has to be enabled in the user's BASIC program, and also the options of the symgology should be taken into account. For example, if the option "Transmit Start/Stop" of Code 39 is selected, a barcode with the value "1234567890" printed in Code 39 would not be able to set the Run-time Engine to the "System Mode" because the barcode will be interpreted as "*1234567890*".

The BASIC Compiler Run-time Engine on 201/510/520 allows the user to configure several system settings or to update the embedded BASIC program. The first item of the setup menu will be shown on the LCD as soon as the master setup barcode/magnetic card is read. The items are listed as follows,

1. Station ID — To setup the station ID of the terminal. Enter 2 digits from the numeric keys. Press "Esc" or "F2" to abort the change of the setting.
2. Com1 Baud — To setup the baud rate of COM1. Press "F1" to select the value, "Esc" or "F2" to abort the change of the setting. Default: 9600.
3. Com1 Parity — To setup the parity of COM1. Press "F1" to select the value, "Esc" or "F2" to abort the change of the setting. Default: None.
4. Com1 Data Bit — To setup the data bit of COM1. Press "F1" to select the value, "Esc" or "F2" to abort the change of the setting. Default: 8.
5. Com1 Flow — To setup the flow control method of COM1. Press "F1" to select the value, "Esc" or "F2" to abort the change of the setting. Default: None.
6. Com3 Baud — To setup the baud rate of COM3. Press "F1" to select the value, "Esc" or "F2" to abort the change of the setting. Default: 9600.
7. Com3 Parity — To setup the parity of COM3. Press "F1" to select the value, "Esc" or "F2" to abort the change of the setting. Default: None.
8. Com3 Data Bit — To setup the data bit of COM3. Press "F1" to select the value, "Esc" or "F2" to abort the change of the setting. Default: 8.

9. Com3 Flow — To setup the flow control method of COM3. Press “F1” to select the value, “Esc” or “F2” to abort the change of the setting. Default: None.

10. Setup Master ID — To setup the ID of the master setup card by scanning the new master setup card. Press “Esc” or “F2” to abort the change of the setting.

11. Basic Download — To download new BASIC program via RS-232 port. The COM port properties are fixed to 38400, no parity, 8 data bits, no handshake.

12. Erase Program — To erase the existing BASIC program.

Press “Enter” key to rotate through the options on the LCD. The user can scan the master setup card again or just turn the unit off and on to reboot the terminal. The new settings will be activated after reboot.

(2) **7xx**

Pressing the 7, 9 and POWER keys simultaneously will enable the 7xx terminals to enter the “System Mode” which provides the following options:

1. Memory
   - Size Information — Includes the SRAM (Data memory) size and FLASH (Program memory) size in kilobytes.
   - Initialize — To initialize the data memory (RAM). Note that the contents of RAM will be cleared after memory initialization.
   - Test — To test the data memory. For 256 KB SRAM, it takes about 15 seconds to complete the test. Note that this is a destructive memory test and the contents of RAM will also be cleared.

2. Setting
   - Clock — To set new date and time.
   - Backlit — To set the intensity of LCD backlight and the duration for staying on. Default: High intensity, the LCD backlightlight will switch off after 20 seconds.
   - Speed — To set CPU running speed. There are five speeds available: *Full speed*, *1/2 speed*, *1/4 speed*, *1/8 speed* and *1/16 speed*; which need about 39mA, 22mA, 12mA, 7mA and 5mA, respectively, during normal operation (without scanning or data transmission). If high-speed operation is not required, selecting low CPU speed will save battery power. Default: Full speed.
   - Auto Off — Set the time required for auto power-off. This value is applied when the unit is idle. If this value is set to zero, this function will be disabled. Default: 10 minutes
   - Power On — There are two possible selections: *Program Resume*, which starts from the program being used during the last session before the last power-off; and *Program Restart*, which starts with a new program. Default: Program Resume
   - Station ID — Set station ID number for the terminal. This ID number is used to identify the terminal when it is transmitting data in a cradle that is connected with other cradles. Default: 1
   - Key Click — To enable / disable key click sound. Default: Enable

3. Reader

- Reading test — To test the reading performance of the scanner. The following are the default enabled symbologies in this mode:
  - Code 39
  - Industrial 25
  - Interleave 25
  - Codabar
  - Code 93
  - Code 128
  - UPCE
  - UPCE with ADDON 2
  - UPCE with ADDON 5
  - EAN8
  - EAN8 with ADDON 2
  - EAN8 with ADDON 5
  - EAN13
  - EAN13 with ADDON 2
  - EAN13 with ADDON 5

  Other symbologies must be enabled through programming.

4. Battery

- Main — To show the voltage of the main battery power.
- Backup — To show the voltage of the backup battery.

5. Test

- Buzzer — To test the buzzer with different Frequency/Duration. Press the ENTER key to start / stop the test.
- LCD & LED — To test LCD display and LED indicator. Press ENTER key to start / stop the test.
- KBD — To test the rubber keys. Press a key and the result will be shown on the LCD display. Note that the FN key of 711 should be used in conjunction with numeric keys.

6. Download

- RS-232 — To download the user program via the RS-232 port. The transmission speed can be up to 115200 bps.
- Docking — To download the user program via communication cradle.
- IR — To download the user program via serial IR transceiver. The transmission speed can be up to 115200 bps.
- IrDA — To download the user program via standard IrDA. The transmission speed can be up to 115200 bps.

7. Version

- Version Info — To show version information, including hardware version, serial number of the terminal, manufacturing date, kernel version, library version and information of the user's application program.

2 Using CipherLab BASIC Compiler

CipherLab BASIC Compiler looks like a traditional Windows environment application that supports file management, text editing, and some other functions to simplify the BASIC program development. There are five menus on the menu bar, i.e., “File”, “Edit”, “Config”, “Compile”, and “Help”, and each menu provides several commands/items. This chapter discusses the function and operation for each command/item.

2.1 File Menu

Six commands are provided on this menu.

(1) **New**

Function: To create a new BASIC program.

Operation: Click “File” on the menu bar and select “New”; or type Ctrl + N; or click the “New” icon on the tool bar.

(2) **Open**

Function: To open an existing BASIC program.

Operation: Click “File” on the menu bar and select “Open”; or type Ctrl + O; or click the “Open” icon on the tool bar.

(3) **Save**

Function: To save the current editing BASIC program.

Operation: Click “File” on the menu bar and select “Save”; or type Ctrl + S; or click the “Save” icon on the tool bar.

(4) **Save As**

Function: To save the current editing BASIC program with a new name.

Operation: Click “File” on the menu bar and select “Save As” to pop-up the “Save As” window. Enter a new file name, and click “Save” button to save this program with the new file name.

(5) **Print**

Function: To print the current editing BASIC program.

Operation: Click “File” on the menu bar and select “Print”; or type Ctrl + P; or click the “Print” icon on the tool bar.

(6) **Exit**

Function: To quit the BASIC Compiler.

Operation: Click “File” on the menu bar and select “Exit”; or type Alt + F4.

2.2 Edit Menu

Seven commands are provided here to facilitate editing of the BASIC source code.

(1) **Undo**

Function: To abort the previous editing command or action.

Operation: Click “Edit” on the menu bar and select “Undo”; or type Ctrl + Z; or click the “Undo” icon on the tool bar.

(2) **Cut**

Function: To cut a paragraph off the text and keep it in the clipboard. The paragraph will be cleared.

Operation: Drag the cursor to select the paragraph to be cut off. This paragraph will turn to reverse colour. Click “Edit” on the menu bar and select “Cut”; or type Ctrl + X; or click the “Cut” icon on the tool bar.

(3) **Copy**

Function: To copy a paragraph from the text and keep it in the clipboard.

Operation: Select the paragraph to be copied. Click “Edit” on the menu bar and select “Copy”; or type Ctrl + C; or click the “Copy” icon on the tool bar.

(4) **Paste**

Function: To paste a paragraph from the clipboard into the text. This paragraph will be inserted into the text.

Operation: Move the cursor to the location where the paragraph will be inserted and click the left button on the mouse then click “Edit” on the menu bar and select “Paste”; or type Ctrl + V; or click the “Paste” icon on the tool bar.

(5) **Delete**

Function: To delete a paragraph from the text. This paragraph will not be kept in the clipboard.

Operation: Select the paragraph to be deleted. Click “Edit” on the menu bar and select “Delete”; or press the Del key.

(6) **Select All**

Function: To select all the contents of the text.

Operation: Click “Edit” on the menu bar and select “Select All”; or type Ctrl + A. All the contents of the text will turn to reverse colour.

(7) **Find**

Function: To find a special letter, symbol, word, or paragraph in the text.

Operation: Click “Edit” on the menu bar and select “Find”; or type Ctrl + F; or click the “Find” icon on the tool bar to pop-up the “Find” window. Enter the key word to be found in the text and then click the “Find” button.

2.3 Config Menu

Seven items are provided for the user to define the system settings.

(1) **Target Machine**

Function: To set the type of the target machine.

Operation: Click “Config” on the menu bar and select “Target Machine”, and then scroll through the items in the “Target Machine” window to set the target machine. The selection of the target machine will affect the number of transaction files, the available baud rate of the COM port, and whether the user can create the DBF files on the Smart-Media Card.

(2) **Master Card ID**

Function: To define the ID of the master setup card.

Operation: Click “Config” on the menu bar and select “Master Card ID” to pop-up the “Master Card ID” window and then type the new card ID in the field. This feature is only valid for fixed terminals (201/510/520).

(3) **Primary COM Port Setting**

Function: To set the properties of the primary COM port.

Operation: Click “Config” on the menu bar and select “Primary COM Port Setting” to pop-up the “Primary COM Port Properties” window. Then select the desired settings for each property.

(4) **Secondary COM Port Setting**

Function: To set the properties of the secondary COM port.

Operation: Click “Config” on the menu bar and select “Secondary COM Port Setting” to pop-up the “Secondary COM Port Properties” window. Then select the desired settings for each property.

(5) **Config Transaction Files**

Function: To define how many transaction files to be used and to define the data length for each transaction file.

Operation: Click “Config” on the menu bar and select “Config Transaction Files” to pop-up the “Config Transaction Files” window. Check the boxes to enable the use of the transaction file and type the data length for each enabled transaction file.

(6) **Create DBF Files**

Function: To define the DBF files to be used and to define IDX files for each DBF file.

Operation: Click “Config” on the menu bar and select “Create DBF Files” to pop-up the “Create DBF Files” window. Type the total record length for each DBF file and define the offset and length for IDX files. (For 720 terminal, the user can choose to create DBF files on the Smart-Media Card.)

(7) **Barcode Setting**

Function: To configure the system parameters pertaining to barcode symbologies and scanner performance.

Operation: Click “Config” on the menu bar and select “Barcode Setting” to pop-up the “Barcode Setting” window. Check the box to enable the decodability of the target terminal to the particular barcode symbology. For the description of each barcode setting, please refer to the Appendix D.

2.4 Compile Menu

Three commands are provided on this menu.

(1) **Syntax checking**

Function: To check the syntax of the BASIC program. Any errors found, will pop-up the “Output” window and display the relevant syntax error messages.

Operation: Click “Compile” on the menu bar and select “Syntax checking”. If there are any syntax errors in the BASIC program, the system will pop-up the “Output” window and show the line numbers and the error messages of all errors found.

(2) **Compile**

Function: To compile the BASIC program. If there are any syntax or compiling errors, an “Output” window will pop-up and display the error messages.

Operation: Click “Compile” on the menu bar and select “Compile”; or click the “Compile” icon on the tool bar. If there are any syntax/compiling errors in the BASIC program, the system will pop-up the “Output” window and show the line numbers and the error messages of the errors found. If the compilation is successful, the message “Build successfully, do you want to download the program?” will be shown on the screen. Click the “Yes” button if you want to download the program. (For downloading operation, please refer to the “Download” command.)

(3) **Download**

Function: To download a compiled BASIC program to the target terminal.

Operation: Click “Compile” on the menu bar and select “Download” to pop-up the “Open” window. Select the BASIC object file (**.syn**) to be downloaded and then click “Open”. Select the correct COM port properties and then click “OK” to download. For 7xx terminal, the user may select to download the BASIC program via Serial IR transceiver.

Please note that the associated system initialisation file (**.ini**) has to be in the same directory as the BASIC object file, otherwise the default system settings will be downloaded instead.

2.5 Help Menu

Three items are provided on this menu.

(1) **Contents**

By selecting this item, Windows Help manager will be invoked and a table of contents for the BASIC online documentation will be displayed. The user may click on any listed item for more detailed information.

(2) **Index**

By selecting this item, the help file index will be shown. User can either type in a string for searching or directly click on the listed index for viewing further information.

(3) **About**

A message box containing ownership declaration and version information will be shown upon selecting this menu item.

3 Basics of CipherLab BASIC Language

3.1 Constants

Constants are the actual values that BASIC uses during execution. There are two types of constants: string and numeric.

3.1.1 String

A string constant is a sequence of up to 255 alphanumeric characters or symbols enclosed in double quotation marks. Examples of string constants are:

a. “Hello”

b. “$20,000.00”

c. “12 Students”

3.1.2 Numeric

Numeric constants are positive or negative numbers. Numeric constants in BASIC cannot contain commas. There are three types of numeric constants that can be used in CipherLab BASIC Compiler:

a. Integer Constants – whole numbers between –32,768 and +32,767. Integer constants do not have decimal points.

b. Real Number Constants – positive or negative real numbers, i.e., numbers that contain decimal points.

c. Long Integer Constants – whole numbers between –2,147,483,648 and +2,147,483,647.

3.2 Variables

Variables are symbols used to represent data items, such as numerical values or character strings that are used in a BASIC program. The value of a variable may be assigned explicitly and can be changed during the execution of a program. Before a variable is assigned a value, its value is assumed to be undefined.

3.2.1 Variable Names and Declaration Characters

Following are the rules for variable names and declaration characters:

a. A variable name must begin with a letter (A through Z).

b. The remaining characters can be letters, numbers, and/ or underscores.

c. The last character can be one of these type declaration characters:

i.	% integer	:	2 bytes	-32,768 to +32,767
ii.	& long	:	4 bytes	-2,147,483,648 to +2,147,483,647
iii.	! real number	:	4 bytes	
iv.	$ string			
v.	nothing (default)	:	2 bytes	-32,768 to +32,767

d. The variable name cannot be a BASIC reserved word.

e. Variable names are not case sensitive.

3.2.2 Array Variables

An array is a group or table of values referenced by the same variable name. Each element in an array is referenced by an array variable that is subscripted with an integer or an integer expression.

a. An array variable name has as many dimensions as there are subscripts in the array. For example,
   i. A(12) would reference a value in a one-dimension array.
   ii. T(2,5) would reference a value in a two-dimension array, and so on.

b. Each element in an array is referenced by an array variable that is subscripted with an integer or an integer expression. For example,

i.	DIM IntegerA%(20)	Declare an integer array with 20 elements.
ii.	DIM SringB$(100)	Declare a string array with 100 elements.
iii.	DIM RealC!(10)	Declare a real number array with 10 elements.
iv.	DIM Tb(5,5)	Declare a two-dimension integer array with 5X5 elements.
v.	ArrayD(i+1, j)	The elements of an array are subscripted with an integer expression.

c. The first element of an array is subscripted with 1.

d. In CipherLab BASIC, the maximum number of dimensions for an array is 2, and up to 32,767 elements per dimension is allowed while compiling.

3.3 Expression and Operators

An expression may be a string or numeric constant, or a variable, or it may combine constants and variables with operators to produce a single value. Operators perform mathematical or logical operations or values. The operators provided by CipherLab BASIC Compiler may be divided into four categories, namely, Assignment Operator, Arithmetic Operators, Relational Operators, and Logical Operators.

3.3.1 Assignment Operator

CipherLab BASIC Compiler supports an assignment operator: "=". For example,

a. Length% = 100

b. PI! = 3.14159

c. Company$ = "Syntech Information Co., Ltd."

3.3.2 Arithmetic Operator

The arithmetic operators are:

Operator	Operation	Sample Expression
^	Exponentiation	A% = 9^3
-	Negation (unary)	A% = -B%
*	Multiplication	A! = B! * C!
\	Division (integer)	A% = B! \ C!
/	Division (real)	A! = B! / C!
+	Addition	A% = B% + C%
-	Subtraction	A% = B% - C%
MOD	Modulo arithmetic	A% = B% MOD C%

3.3.3 Relational Operator

Relational operators are used to compare two values. The result of the comparison is either "True" or "False." This result may then be used to make a decision regarding program flow.

Operator	Operation	Sample Expression
=	Equality	A% = B%
<>	Inequality	A% <> B%
><	Inequality	A! >< B!
>	Greater than	A% > B!
<	Less than	A! < B!
>=	Greater than or equal to	A% >= B%
<=	Less than or equal to	A% <= B%

3.3.4 Logical Operator

Logical operators perform tests on multiple relations and Boolean operations. The logical operator returns a bit-wise result which is either “True” (not zero) or “False” (zero). In an expression, logical operations are performed after arithmetic and relational operations.

Operator	Operation	Sample Expression
NOT	Logical negation	IF NOT (A% = B%)
AND	Logical and	IF (A% = B%) AND (C% = D%)
OR	Inclusive or	IF (A% = B%) OR (C% = D%)
XOR	Exclusive or	IF (A% = B%) XOR (C% = D%)

3.4 Operator Precedence

The precedence of BASIC operators affects the evaluation of operands in expressions. Expressions with higher precedence operators are evaluated first. The precedence of BASIC operators is listed below in order of precedence from highest to lowest. Where several operators appear together, they have equal precedence.

Type of Operation	Symbol
Exponentiation	^
Multiplicative	*, /, \, MOD
Additive	+, -
Relational	=, <>, >, <, >=, <=
Logical	AND, NOT, OR, XOR
Assignment	=

3.5 Labels

Line labels are used to represent some special lines in the BASIC program. They can be either integer numbers or characters strings. A valid integer number for the line label is in the range of 1 to 32,767. A character string label can have up to 49 characters. If the string label has more than 49 characters, it will be truncated to 49 characters long. A character string label that precedes a program line must have a colon, "**:**", between label and the program, but an integer label doesn't. For example,

```
        GOTO 100
        …
100     PRINT “This is an integer label.”
        …
        GOTO Label2
        …
Label2: PRINT “This is a character string label.”
```

3.6 Subroutines

A subroutine is a set of instructions given a particular name or a line label. Users can simplify their programming by breaking programs into smaller logical subroutines. A subroutine will be executed when called by a GOSUB command. For example,

```
        ON KEY(1) GOSUB KeyF1
        …
KeyF1:
        PRINT “F1 is pressed.”
        RETURN
```

The command RETURN marks the end of the subroutine and tells the processor to return to the caller. A subroutine has to be appended at the end of the main BASIC program. A subroutine can be defined with or without a pair of brackets. For example,

```
SUB Subroutine1( )
        …
        PRINT “Subroutine1 is executed.”
        END SUB
…
SUB Subroutine2
        …
        PRINT “Subroutine2 is executed.”
        END SUB
```

Since all the variables in the CipherLab BASIC program are treated as global variables, passing arguments to subroutines is meaningless and enclosing arguments in the brackets of the subroutines will lead a syntax error while compiling.

A subroutine in BASIC can be recursive, meaning it can call itself or other subroutines that in turn call the first subroutine. The following sample program contains a recursive subroutine--*Factorial*, to calculate the value of n! (“n factorial”).

```
        PRINT “Please enter a number (1 – 13): “
        INPUT N%
        FactResult! = 1
        Fact% = N%
        GOSUB Factorial
        PRINT N%, “! = ”, FactResult!
Loop:
        GOTO Loop


Factorial:
        IF Fact% < 1 THEN RETURN
        FactResult! = FactResult! * Fact%
        Fact% = Fact% -1
        GOSUB Factorial
        RETURN
```

3.7 Programming Style

Following are the guidelines used in writing programs in this manual and also in the sample programs. These guidelines are recommended for program readability, but they are not compulsory.

- Reserved words and symbolic constants appear in uppercase letters:

```
PRINT  “510 Demo Program”
BEEP (800, 30, 0, 5, 800, 15, 0, 5, 800, 15)
```

- Variable names are in lowercase with an initial capital letter; if variable names are combined with more than one part, other capital letters may be used to make it easier to read:

```
ProcessFlag% = 0
Temp$ = GET_RECORD$(3,1)
```

- Line labels are used instead of line numbers.

```
ON READER(2) GOSUB GetSlotReader
```

4 BASIC Commands

This chapter provides detailed descriptions for the commands supported by the CipherLab BASIC Compiler. Besides the commands commonly used in traditional versions of BASIC, several additional commands that deal with specific hardware features of the CipherLab terminals are supported. These commands are called within the user's BASIC programs to perform a wide variety of tasks, including communications, LCD, buzzer, scanner, file manipulation, etc. They are categorized and described in this chapter by their functions or the resources they work on.

Some commands are postfixed with a dollar sign, $, that means a string is returned with the command. The compiler will accept these commands with or without the dollar sign. However, the dollar sign will be postfixed to these commands in this manual and the sample program.

The description for each BASIC command consists of four parts, **Purpose**, **Syntax**, **Remarks**, and **Usage**, which are listed below.

(1) **Purpose**
Briefly explains the purpose of the command.

(2) **Syntax**
Describes the command syntax according to the following conventions:

a. CAPS
BASIC keywords are indicated by capital letters.

b. Itatlics
Items in *Itatlics* represent variable information to be supplied by the user.

c. []
Square brackets indicate optional parameters.

d. {}
Braces indicate an item may be repeated as many times as necessary.

e. |
Vertical bar indicates alternative option.

(3) **Remarks**
Supplies additional information in detail regarding correct command usage.

(4) **Usage**
Illustrates various ways of using the statement and highlights, applicable and unusual modes of operation.

The types of terminals that support the specified BASIC command are listed to the right of the title bar of the command.

4.1 General Commands

This section describes the commands that are not confined to any specific hardware features.

ABS	**201**	**510**	**520**	**711**	**720**

Purpose The ABS function returns the absolute value of a numeric expression.

Syntax *A* = ABS (*N*)

Remarks "*A*" is a numeric variable to be assigned with the absolute value of a numeric expression.

"*N*" is a numeric expression, its result can be interger or real number.

Usage TimeDifference% = ABS (Time1% - Time2%)

DIM	**201**	**510**	**520**	**711**	**720**

Purpose To specify the maximum values of variable subscripts and to allocate storage accordingly.

Syntax DIM *Array* (*range* {,*range*}) {, *Array*(*range* {,*range*})}

Remarks *Array* is an array variable.

range can be an integer or an integer expression.

The DIM statement sets all the elements of the specified arrays to an initial value of zero or empty string.

Note the maximum allowable number of dimensions for an array is 2.

Usage DIM A(10), B%(20), C$(30,10)

GOTO	**201**	**510**	**520**	**711**	**720**

Purpose The GOTO statement branches unconditionally out of the normal program sequence to a specified line number or line label.

Syntax GOTO *LineNumber* | *LineLabel*

Remarks "*LineNumber*" is the interger number in front of a line.

"*LineLabel*" is the string label of a line to branch to.

Usage Loop:

GOTO Loop

GOSUB	**201**	**510**	**520**	**711**	**720**

Purpose To call the specified subroutine.

Syntax GOSUB *SubName* | *SubLabel*

Remarks "*SubName*" is the name of a subroutine.

"*SubLabel*" is the line label of a subroutine.

Usage

GOSUB DoIt

…

```
    GOSUB Done
…
SUB DoIt( )
    PRINT "Now I've done it!"
END SUB
…
Done:
    PRINT "Now you've done it."
RETURN
```

INT	**201**	**510**	**520**	**711**	**720**

Purpose The INT function returns the largest integer that is less than or equal to the given numeric expression.

Syntax *A*% = INT(*N*)

Remarks "*A%*" is an integer variable to be assigned with the result.
"*N*" is a numeric expression.

Usage
```
A% = INT(-2.86)          ' A% = -3
B% = INT(2.86)           ' B% = 2
```

REM	**201**	**510**	**520**	**711**	**720**

Purpose To insert explanatory remarks in a program.

Syntax
REM *remark*
' *remark*

Remarks *remark* may be any sequence of characters.
The BASIC compiler will ignore whatever follows the REM or ' until end of the line.

Usage
```
REM This is a comment.
' This is a comment too.
```

SET_PRECISION	**201**	**510**	**520**	**711**	**720**

Purpose To set the precision of the decimal points for printing real number expressions.

Syntax SET_PRECISION (*N%*)

Remarks "*N%*" is a numeric expression in the range of 0 to 6.
The precision is set to two digits by default.

Usage
```
PI! = 3.14159
PRINT "PI = ", PI!       ' result: PI = 3.14 (by default)
SET_PRECISION(6)
PRINT "PI = ", PI!       ' result: PI = 3.141590
SET_PRECISION(2)
PRINT "PI = ", PI!       ' result: PI = 3.14
```

SGN	201	510	520	711	720

Purpose To return an indication of the mathematical sign (+ or -) of the given numeric expression.

Syntax *A*% = SGN(*N*)

Remarks "*A%*" is an integer variable to be assigned with the result.

"*N*" is a numeric expression.

If *N* > 0, SGN(*N*) returns 1.

If *N* = 0, SGN(*N*) returns 0.

If *N* < 0, SGN(*N*) returns -1.

Usage

```
A% = SGN(100)     ' A% = 1
B% = SGN(-1.5)    ' B% = -1
```

4.2 Commands for Decision Structures

Based on the value of an expression, decision structures cause a program to take one of the following two actions:

(1) Execute one of several alternative statements within the decision structure itself.

(2) Branch to another part of the program outside the decision structure.

In CipherLab BASIC, decision-making is handled by the IF…THEN…[ELSE…][ENDIF] and ON…GOSUB|GOTO…statement.

The IF…THEN…[ELSE…][ENDIF] statement can be used anywhere the ON…GOSUB|GOTO… statement can be used. The major difference between the two is that ON…GOSUB|GOTO… evaluates a single expression, and then executes different statements or branches to different parts of the program based on the result. In contrast, a block IF…THEN…[ELSE…][ENDIF] can evaluate completely different expressions. Moreover, the expression given in the ON *expression* GOSUB|GOTO… statement must evaluate to a number within the range 1 to 255, while the expression in IF…THEN…[ELSE…][ENDIF] statement can only evaluate to either a TRUE or FALSE condition.

IF…THEN…[ELSE…] **201 510 520 711 720**

Purpose To provide a decision structure for single-line conditional execution.

Syntax IF *condition* THEN *action1* [ELSE *action2*]

Remarks "*condition*" is a logical expression.
"*action*" is a BASIC statement.

Usage
```
IF Data1% > Data2% THEN Temp% = Data1% ELSE Temp% = Data2%
```

IF…THEN…{ELSE IF…}[ELSE…]END IF **201 510 520 711 720**

Purpose To provide a decision structure for a multiple-line conditional excution.

Syntax
```
IF condition1 THEN
    statementblock1
{ELSE IFcondition2 THEN
    Statementblock2}
[ELSE
    StatementblockN]
END IF
```

Remarks "*condition*" is a logical expression.
"*Statementblock*" can be multiple lines of BASIC statements.

Usage
```
IF LEFT$(String1$,1) = "A" THEN
    PRINT "String1 is leaded by A."
ELSE IF LEFT$(String1$,1) = "B" THEN
    PRINT "String1 is leaded by B."
ELSE
    PRINT "String1 is not leaded by A nor B."
END IF
```

ON… GOTO…

201 510 520 711 720

Purpose Branches to one of several specified LineLabels depending on the value of an expression.

Syntax ON *N* GOTO *LineLabel* {, *LineLabel*}

Remarks "*N*" is a numeric expression which is rounded to an integer. The value of *N* determines which line label in the list will be used for branching. If the value of *N* is 0, or greater than the number of line labels listed, the interpreter will continue with the next executable statement.

"*LineLabel*" is the label of a line to branch to.

Usage

```
PRINT "Input a number (1-9): "
INPUT Num%
CLS
ON Num% GOTO 100, 100, 100, 200, 200, 300, 400, 400, 400
…
100
    PRINT "Number 1-3 is inputed."
    GOTO 500
200
    PRINT "Number 4-5 is inputed."
    GOTO 500
300
    PRINT "6 is inputed."
    GOTO 500
400
    PRINT "Number 7-9 is inputed."
500
…
```

ON…GOSUB…

201 510 520 711 720

Purpose Call one of the several specified subroutines depending on the value of the expression.

Syntax ON *N* GOSUB *SubLabel* {, *SubLabel*}

Remarks "*N*" is a numeric expression that is rounded to an integer. The value of *N* determines which subroutine to be called. If the value of *N* is 0, or greater than the number of routines listed, the interpreter will continue with the next executable statement.

"*SubLabel*" is the label or name of a subroutine to be called.

Usage

```
PRINT "Input a number (1-9): "
INPUT Num%
CLS
ON Num% GOSUB 100, 100, 100, 200, 200, 300, 400, 400, 400
…
100
    PRINT "Number 1-3 is inputed."
    RETURN
200
```

```
    PRINT “Number 4-5 is inputed.”
    RETURN
300
    PRINT “6 is inputed.”
    RETURN
400
    PRINT “Number 7-9 is inputed.”
    RETURN
…
```

IF…THEN…END IF **201 510 520 711 720**

Purpose To provide a decision structure for a conditional excution with multipule lines of actions.

Syntax

```
IF condition1 THEN
    action1
    action2
    …
END IF
```

Remarks “*condition*“ is a logical expression.
“*action*“ is a BASIC statement.

Usage

```
IF Data1% > Large% THEN
    BEEP(800,30)
    Large% = Data1%
    PRINT “Current Largest Number is “, Data1%
END IF
```

4.3 Commands for Looping Structures

Looping structures repeat a block of statements, either for a specified number of times or until a certain condition is matched. In CipherLab BASIC, two kinds of looping structures, FOR…NEXT and WHILE…WEND can be used. Command EXIT can be used for an alternative way to exit from both FOR…NEXT and WHILE…WEND loops. Both FOR…NEXT and WHILE…WEND statements can be nested up to 10 levels.

FOR…NEXT	**201**	**510**	**520**	**711**	**720**

Purpose To repeat the execution of a block of statements for a specified number of times.

Syntax

FOR *N%* = *startvalue* TO *endvalue* [STEP *step*]
    [*Statement Block*]
NEXT [*N%*]

Remarks

"*N%*" is an integer variable to be used as a loop counter.

"*startvalue*" is a numeric expression which is the initial value for the loop counter.

"*endvalue*" is a numeric expression which is the final value for the loop counter.

"*step*" is a numeric expression to be used as an increment/decrement of the loop counter. The "step" is 1 by default.

If the loop counter ever reaches or beyond the endvalue, the program execution continues to the statement following the NEXT statement. The Statement block will be executed again otherwise.

Usage

```
DataCount% = TRANSACTION_COUNT
FOR Counter% = 1 TO DataCount%
    Data$ = GET_TRANSACTION_DATA$(Counter%)
    WRITE_COM(1,Data$)
NEXT
```

EXIT	**201**	**510**	**520**	**711**	**720**

Purpose To provide an alternative exit for looping structures, such as FOR…NEXT, and WHILE…WEND statements.

Syntax EXIT

Remarks EXIT can appear anywhere within the loop statement.

Usage

```
DataCount% = TRANSACTION_COUNT
FOR Counter% = 1 TO DataCount%
    Data$ = GET_TRANSACTION_DATA$(Counter%)
    HostCommand$ = READ_COM(1)
    IF HostCommand$ = "STOP" THEN EXIT
    WRITE_COM(1,Data$)
NEXT
```

WHILE…WEND	201	510	520	711	720

Purpose To repeat the execution of a block of statements while a certain condition is TRUE.

Syntax

WHILE *condition*
    [*Statement Block*]
WEND

Remarks If the "*condition*" is true, loop statements are executed until the WEND statement is encountered. Then the program execution returns to the WHILE statement and checks the condition again. If it is still true, the process will be repeated. Otherwise, the execution continues with the statement following the WEND statement.

Usage

```
WHILE TRANSACTION_COUNT > 0
    Data$ = GET_TRANSACTION_DATA$(1)
    WRITE_COM(1,Data$)
    DEL_TRANSACTION_DATA(1)
WEND
```

4.4 Commands for String Processing

This section describes BASIC commands used to manipulate sequences of ASCII characters known as strings. In CipherLab BASIC, strings are always variable length, from null to a maximum of 255.

4.4.1 Combining Strings

Two strings can be combined with the plus, +, operator. The string following the plus operator is appended to the string preceding the plus operator. For example,

```
…
Data$ = DATE$ + TIME$ + EmployeeID$
SAVE_TRANSACTION(Data$)
…
```

4.4.2 Comparing Strings

Two strings can be compared with the relational operators, cf. 3.3.3. A single character is greater than another character if its ASCII value is greater. For example, the ASCII value of the letter “B” is greater than the ASCII value of the letter “A,” so the expression “B” > “A” is true.

When comparing two strings, BASIC looks at the ASCII values of corresponding characters. The first character where the two strings differ determines the alphabetical order of the strings. For example, the strings “aaabaa” and “aaaaaaaa” are the same up to the fourth character in each, “b” and “a”. Since the ASCII value of “b” is larger than that of “a”, the expression “aaabaa” > “aaaaaaaa” is true.

If there is no difference between corresponding characters of two strings and they are the same length, then the two strings are equal. If there is no difference between corresponding characters of two strings, but one of the strings is longer, the longer string is greater than the shorter string. For example, “abc” = “abc” and “aaaaaaaa” > “aaaaa” are both true expressions.

Leading and trailing blank spaces are significant in string comparisons. For example, the string “ abc” is less than the string “abc”, since a blank space is less than an “a”; on the other hand, the string “abc “ is greater than the string “abc”.

4.4.3 Getting the Length of a String

LEN	**201**	**510**	**520**	**711**	**720**

Purpose To return the length of a string.

Syntax *A*% = LEN(*X$*)

Remarks “*A%*” is an integer variable to be assigned with the result.
“*X$*” may be a string variables, string expressions, or string constants.
Non-printing characters and blanks are counted.

Usage

```
String1$ = “abcde “
A% = LEN(String1$)
‘ A% = 6, including the blank
```

4.4.4 Searching for Strings

Seraching for a string inside another one is one of the most common string-processing tasks. INSTR is provided for this task.

INSTR	201	510	520	711	720

Purpose To search if one string exists inside another one.

Syntax *A*% = INSTR ([*N%*], *X$*, *Y$*)

Remarks "*A%*" is an integer variable to be assigned with the result.

"*N%*" is a numeric expression in the range of 1 to 255. Optional offset *N* sets the position for starting the search.

"*X$*", "*Y$* " may be a string variables, string expressions, or string constants.

If *Y$* is found in *X$*, INSTR returns the position of the first occurrence of *Y$* in *X$*, from the starting point.

If *N* is larger than the length of *X$*, or if *X$* is null, or if *Y$* cannot be found, INSTR returns 0.

If *Y$* is null, INSTR returns *N* (or 1 if *N* is not specified).

Usage

```
String1$ = "11025John Thomas,Accounting Manager"
String2$ = ","
EmployeeName$ = MID$(String1$, 6, INSTR(String1$, String2$) - 6)
' The employee's name starts at the sixth character.
```

4.4.5 Retrieving Part of Strings

Several commands are provided to take strings apart, returning pieces of a string, either from the left side, the right side, or the middle of the target string.

LEFT$	201	510	520	711	720

Purpose To retrieve a given number of characters from the left side of the target string.

Syntax *A*$ = LEFT$(*X$*, *N%*)

Remarks "*A$*" is a string variable to be assigned with the result.

"*N%*" is a numeric expression in the range 0 to 255.

"*X$*" may be a string variables, string expressions, or string constants.

If *N* is larger than the length of *X$*, the entire string (*X$*) will be returned.

If *N* is zero, the null string (with length 0) is returned.

Usage

```
String1$ = "11025John Thomas,Accounting Manager"
EmployeeID$ = LEFT$(String1$, 5)
```

MID$	201	510	520	711	720

Purpose To retrieve a given number of characters from anywhere in the target string.

Syntax *A*$ = MID$(*X$*, *N%*[, *M%*])

Remarks "*A$*" is a string variable to be assigned with the result.

"*N%*" and "*M%*" are numeric expressions in the range 0 to 255.

"*X$*" may be a string variable, string expression, or string constant.

MID$ function returns a string of length *M* characters from *X$* beginning with the *N*th character.

If *M* is omitted, or if there are fewer than *M* characters to the right of the *N*th character, all rightmost characters beginning with the *N*th character are returned.

If *M* is equal to zero, or if *N* is greater than the length of *X$*, then MID$ returns a null string.

Usage

```
String1$ = "11025John Thomas,Accounting Manager"
String2$ = ","
EmployeeName$ = MID$(String1$, 6, INSTR(String1$, String2$) - 6)
' The employee's name starts at the sixth character.
```

RIGHT$	**201**	**510**	**520**	**711**	**720**

Purpose To retrieve a given number of characters from the right side of the target string.

Syntax *A*$ = RIGHT$(*X$*, *N%*)

Remarks "*A$*" is a string variable to be assigned with the result.

"*N%*" is a numeric expression in the range 0 to 255.

"*X$*" may be a string variable, string expression, or string constant.

If *N* is larger than the length of *X$*, the entire string (*X$*) will be returned.

If *N* is zero, the null string (with length 0) is returned.

Usage

```
String1$ = "11025 John Thomas, Accounting Manager"
String2$ = ","
Title$ = RIGHT$(String1$, LEN(String1$) - INSTR(String1$, String2$))
```

TRIM_LEFT$	**201**	**510**	**520**	**711**	**720**

Purpose To return a copy of a string with leading blank spaces stripped away.

Syntax *A*$ = TRIM_LEFT$(*X$*)

Remarks "*A$*" is a string variable to be assigned with the result.

"*X$*" is a string that may contain some space characters at the beginning.

Usage

```
S1$ = TRIM_LEFT$("   Hello World!  ")        ' S1$ = "Hello World!  "
```

TRIM_RIGHT$	**201**	**510**	**520**	**711**	**720**

Purpose To return a copy of a string with trailing blank spaces stripped away.

Syntax *A*$ = TRIM_RIGHT$(*X$*)

Remarks "*A$*" is a string variable to be assigned with the result.

"*X$*" is a string that may contain some space characters at the end.

Usage

```
S2$ = TRIM_RIGHT$("  Hello World!   ")       ' S2$ = "  Hello World!"
```

4.4.6 Converting for Strings

Several commands are available for converting strings to uppercase or lowercase letters, and converting strings to numbers, and numbers to strings.

ASC	**201**	**510**	**520**	**711**	**720**

Purpose To return the decimal value of the ASCII code of the first character of a given string.

Syntax *A*% = ASC(*X$*)

Remarks "*A%*" is an integer variable to be assigned with the result.
"*X$*" is a character string.

Usage
```
A% = ASC("John Thomas")
' A% = 74
```

CHR$	**201**	**510**	**520**	**711**	**720**

Purpose To return the character with the given ASCII value.

Syntax *A$* = CHR$(*N%*)

Remarks "*A$*" is a string variable to be assigned with the result.
"*N%*" is a numeric expression in the range of 0 to 255.

Usage
```
A$ = CHR$(65)
' A$ = "A"
```

HEX$	**201**	**510**	**520**	**711**	**720**

Purpose Returns a string which represents the hexadecimal value (base 16) of the decimal argument.

Syntax *A$* = HEX$(*N%*)

Remarks "*A$*" is a string variable to be assigned with the result.
"*N%*" is a numeric expression in the range 0 to 2,147,483,647.
"*N%*" is rounded to an integer before HEX$(*N%*) is evaluated.

Usage
```
A$ = HEX$(140)          ' A$ = "8C"
```

LCASE$	**201**	**510**	**520**	**711**	**720**

Purpose To return a copy of a string in which all uppercase letters will be converted to lowercase letters.

Syntax *A$* = LCASE$(*X$*)

Remarks "*A$*" is a string variable to be assigned with the result.
"*X$*" may be a string variable, string expression, or string constant.

Usage
```
String1$ = "John Thomas"
String2$ = LCASE$(String1$)          ' String2$ = "john thomas"
```

OCT$	201	510	520	711	720

Purpose To convert a decimal numeric expression to a string that represent the value of the numeric expression in octal notation.

Syntax *A$* = OCT$(*N%*)

Remarks "*A$*" is a string variable to be assigned with the result.
"*N%*" is a numeric expression in the range 0 to 2,147,483,647.
"*N%*" is rounded to an integer before OCT$(*N%*) is evaluated.

Usage
```
A$ = OCT$(24)                ' A$ = "30"
```

STR$	201	510	520	711	720

Purpose To convert a numeric expression to a string.

Syntax *A$* = STR$(*N%*)

Remarks "*A$*" is a string variable to be assigned with the result.
"*N%*" is a numeric expression.

Usage
```
StationID$ = STR$(GET_STATION_ID)
```

UCASE$	201	510	520	711	720

Purpose To return a copy of a string in which all lowercase letters will be converted to uppercase letters.

Syntax *A$* = UCASE$(*X$*)

Remarks "*A$*" is a string variable to be assigned with the result.
"*X$*" may be a string variable, string expression, or string constant.

Usage
```
String1$ = "John Thomas"
String2$ = UCASE$(String1$)      ' String2$ = "JOHN THOMAS"
```

VAL	201	510	520	711	720

Purpose To return the numeric value of a string expression in long integer form.

Syntax *A&* = VAL(*X$*)

Remarks "*A&*" is an integer / long integer variable to be assigned with the result.
"*X$*" is a string that is comprised of numeric characters. If the first character is not numeric, VAL function returns 0. The VAL function will strip leading blanks, tabs, and linefeeds from the argument string. The return numeric value is in range of –2,147,483,648 to 2,147,483,647.

Usage

```
ON HOUR_SHARP GOSUB OnHourAlarm
…
OnHourAlarm:
    Hour% = VAL(LEFT$(TIME$,2))
    FOR Counter% = 1 TO Hour%
            BEEP(800,50)
            WAIT(200)
    NEXT
    RETURN
```

VALR	~~201~~	~~510~~	~~520~~	**711**	**720**

Purpose To convert a string expression to a real number.

Syntax *A!* = VALR (*X$*)

Remarks "*A!*" is a real number variable to be assigned with the result.

"*X$*" is a string that is comprised of numeric characters. The precision of the converted result is governed by the "SET_PRECISION" function.

Usage

```
A! = VALR ("123.45")
PRINT  "A = ", A!                    REM   A = 123.45
…..
```

4.4.7 Creating Strings of Repeating Characters

STRING$	**201**	**510**	**520**	**711**	**720**

Purpose To return a string containing the specified number of the requested character.

Syntax *A$* = STRING$(*N%*, *X$*)

*A$* = STRING$(*N%*, *J%*)

Remarks "*A$*" is a string variable to be assigned with the result.

"*N%*" is a numeric expression in the range of 0 to 255 indicating the number of the character.

"*J%*" is a numeric expression in the range of 0 to 255 indicating the ASCII code of a character.

"*X$*" may be a string variable or string constant.

Usage

```
IDX_LENGTH% = 20
Data$ = Name$ + STRING$(IDX_LENGTH% - LEN(Name$), " ")
ADD_RECORD$(1, Data$)
' Padding with space if the length of Name$ is less than IDX_LENGTH%.
```

4.5 Commands for Event Trapping

An event is an action recognized by the terminal, such as a function keystroke is detected (KEY event), a signal is received from the serial port (COM event), and so on. There are two ways to detect the occurrence of an event and reroute program control to an appropriate subroutine: polling and trapping.

With event polling, the BASIC program explicitly checks for any events that happen at particular points in its execution. For example, the following statements cause the program to loop back and forth until any key pressed by the user:

```
Loop:
        KeyData$ = INKEY$
        IF KeyData$ = “” THEN GOTO Loop
        …
```

Polling is useful when the occurrence of the event is predictable in the flow of the program. But if the time of the occurrence of the event is not predictable, trapping becomes the better alternative because the program will not be paused by the looping statements. For example, the following statements cause the program rerouting to the Key_F1 subroutine when the key F1 is pressed at anytime.

```
        ON KEY(1) GOSUB Key_F1
        …
Key_F1:
        …
```

This section describes the different events the CipherLab BASIC can trap and the related commands.

4.5.1 Event Triggers

There are 10 different events that can be trapped. They are listed and described as follows,

(1) COM Event: a signal is received from the COM port.

(2) DIGIN Event: a digital input port changes its state.

(3) INQUIRY Event: an inquiry is sent to the master station.

(4) ESC Event: the key ESC is pressed.

(5) HOUR_SHARP Event: the system time is on the hour.

(6) KEY Event: a function key is pressed.

(7) MINUTE_SHARP Event: the system time is on the minute.

(8) NET Event: a net command is received from the RS485 port.

(9) READER Event: a barcode or magnetic-card data is decoded

(10) TIMER Event: an activated timer is time-out.

OFF ALL	201	510	520	711	720

Purpose To disabled all the event triggers.

Syntax OFF ALL

Remarks All the event triggers will be disabled. To resume the event trigger, the command ON *event* GOSUB… has to be called.

Usage

```
ON READER(1) GOSUB BcrData_1
ON READER(2) GOSUB BcrData_2
ON KEY(1) GOSUB KeyData_1
…
IF BACKUP_BATTERY < BATTERY_LOW% THEN
    OFF ALL
    BEEP(2000,30)
    CLS
    PRINT “Backup Battery needs to be replaced!”
    Loop:
        GOTO Loop
END IF
…
```

OFF COM	201	510	520	711	720

Purpose To terminate the “COM Event Trigger” which executes a specific subroutine when data is received from the COM ports.

Syntax OFF COM(*N%*)

Remarks “*N%*” is a positive integer indicates the COM port. For fixed terminals 201, 510, and 520, it can be 1 or 3. For portable terminals 711 and 720, it can be 1 or 2.

Usage

```
    ON COM(1) GOSUB HostCommand
    …
HostCommand_1:
    OFF COM(1)          REM disable the trapping during data processing
    …
    ON COM(1) GOSUB HostCommand
    RETURN
```

OFF DIGIN	201	510	520	~~711~~	~~720~~

Purpose To terminate the “DIGIN Event Trigger” which executes a specific subroutine when the specified digital input changes its state.

Syntax OFF DIGIN(*number%*)

Remarks “*number%*” is a positive integer indicates the digital input port. For 201 and 510, it can be 1 or 2. For 520, it can be from 1 to 4.

Usage

```
…
ON DIGIN(1) GOSUB DI_1
…
DI_1:
  CLS
  PRINT “DI(1) is changed.“
  OFF DIGIN(1)
  RETURN
```

OFF ESC	**201**	**510**	**520**	**711**	**720**

Purpose To terminate the “ESC Event Trigger” which executes a specific subroutine when the key ESC is pressed.

Syntax OFF ESC

Remarks

Usage

```
ON ESC GOSUB Key_Esc
…
Key_Esc:
  OFF ESC
  …
  ON ESC GOSUB Key_Esc
  RETURN
```

OFF HOUR_SHARP	**201**	**510**	**520**	**711**	**720**

Purpose To terminate the “HOUR_SHARP Event Trigger” which executes a specific subroutine when the system time is on the hour.

Syntax OFF HOUR_SHARP

Remarks

Usage OFF HOUR_SHARP

OFF ENQUIRY	**201**	**510**	**520**	~~711~~	~~720~~

Purpose To terminate the “INQUIRY Event Trigger” which executes a specific subroutine when the master station is inquired by a slave station.

Syntax OFF ENQUIRY

Remarks Please refer to section 4.16.2 for the RS485 related commands.

Usage

```
ON ENQUIRY GOSUB Inq
…
Inq:
  OFF ENQUIRY
  Inq$ = GET_INQUIRY$
  …
  ON ENQUIRY GOSUB Inq
  RETURN
```

OFF KEY

201 510 520 711 720

Purpose To terminate the “FUNCTION KEY Event Trigger” which executes a specific subroutine when a function key is pressed.

Syntax OFF KEY(*number%*)

Remarks “*number%*” is a positive integer between 1 to 12. It indicates the function key of the keyboard.

Usage

```
ON KEY(1) GOSUB On_Shift
ON KEY(2) GOSUB Off_Shift
…
On_Shift:
  OFF KEY
  Mode$ = “IN”
  GOSUB Process
  ON KEY(1) GOSUB On_Shift
  RETURN
  …
```

OFF MINUTE_SHARP

201 510 520 711 720

Purpose To terminate the “MINUTE_SHARP Event Trigger” which executes a specific subroutine when the system time is on the minute.

Syntax OFF MINUTE_SHARP

Remarks

Usage OFF MINUTE_SHARP

OFF NET

201 510 520 ~~711~~ ~~720~~

Purpose To terminate the “NET Event Trigger” which executes a specific subroutine when data is received from the RS485 port.

Syntax OFF NET

Remarks Please refer to section 4.16.2 for the use of the RS485 commands.

Usage

```
ON NET GOSUB NetCommand
…
NetCommand:
  OFF NET
  NetCmd$ = READ_NET$
  …
  …
  ON NET GOSUB NetCommand
  RETURN
```

OFF READER	201	510	520	711	720

Purpose To terminate the "READER Event Trigger" which executes a specific subroutine when data is received from the reader ports.

Syntax OFF READER(*N%*)

Remarks "*N%*" may be 1 or 2 for terminals 201, 510, and 520.

"*N%*" is 1 for terminals 711and 720.

Usage

```
ON READER(1) GOSUB BcrData_1
…
BcrData_1:
  OFF READER(1)
  BEEP(2000,5)
  Data$ = GET_READER_DATA$(1)
  CLS
  PRINT Data$
  …
```

OFF TIMER	201	510	520	711	720

Purpose To terminate the "TIMER Event Trigger" which was specified by the ON TIMER… GOSUB… command.

Syntax OFF TIMER(*N%*)

Remarks "*N%*" is an integer between 1 to 5. It indicates the number of the timer that was specified by the user.

Usage

```
ON TIMER(1,200) GOSUB ClearScreen   ' TIMER(1) = 2 sec
…
ClearScreen:
  OFF TIMER(1)
  CLS
  RETURN
```

ON COM… GOSUB…	201	510	520	711	720

Purpose To activate the "COM Event Trigger" which executes a specific subroutine when data is received from the COM ports.

Syntax ON COM(*N%*) GOSUB *SubLabel*

Remarks "*N%*" is a positive integer indicates the COM port. For fixed terminals 201, 510, and 520, it can be 1 or 3. For portable terminals 711 and 720, it can be 1 or 2.

"*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Usage

```
ON COM(1) GOSUB HostCommand
…
HostCommand_1:
  OFF COM(1)
  …
  ON COM(1) GOSUB HostCommand
  RETURN
```

ON DIGIN… GOSUB…

201 510 520 ~~711~~ ~~720~~

Purpose To activate the “DIGIN Event Trigger” which executes a specific subroutine when a digital input changes its state.

Syntax ON DIGIN(*number%*) GOSUB *SubLabel*

Remarks “*number*” is a positive integer indicates the digital input port. For 201 and 510, it can be 1 or 2. For 520, it can be from 1 to 4.

“*SubLabel*“ is the name or line label of the subroutine to be called when the event is triggered.

Usage

```
…
ON DIGIN(1) GOSUB DI_1
…
DI_1:
  CLS
  PRINT “DI(1) is changed.“
  OFF DIGIN(1)
  RETURN
```

ON ESC GOSUB…

201 510 520 711 720

Purpose To activate the “ESC Event Trigger” which executes a specific subroutine when the key ESC is pressed.

Syntax ON ESC GOSUB *SubLabel*

Remarks “*SubLabel*“ is the name or line label of the subroutine to be called when the event is triggered.

Usage

```
ON ESC GOSUB Key_Esc
…
Key_Esc:
  OFF ESC
  …
  ON ESC GOSUB Key_Esc
  RETURN
```

ON HOUR_SHARP GOSUB… **201 510 520 711 720**

Purpose To activate the "HOUR_SHARP Event Trigger" which executes a specific subroutine when the system time is on the hour.

Syntax ON HOUR_SHARP GOSUB *SubLabel*

Remarks "*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Usage

```
…
ON HOUR_SHARP GOSUB OnHourAlarm
…
OnHourAlarm:
  CurrentTime$ = TIME$
  Hour% = VAL(LEFT$(CurrentTime$,2))
  FOR I = 1 TO Hour%
    BEEP(800,10,0,10)
    WAIT(100)
  NEXT
  RETURN
```

ON ENQUIRY GOSUB… **201 510 520** ~~711~~ ~~720~~

Purpose To activate the "INQUIRY Event Trigger" which executes a specific subroutine when the master station is inquired by a slave station.

Syntax ON ENQUIRY GOSUB *SubLabel*

Remarks "*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Please refer to 4.16.2 for the RS-485 related commands.

Usage

```
ON ENQUIRY GOSUB Inq
…
Inq:
  OFF ENQUIRY
  Inq$ = GET_INQUIRY$
  …
  ON ENQUIRY GOSUB Inq
  RETURN
```

ON KEY… GOSUB… **201 510 520 711 720**

Purpose To activate the "FUNCTION KEY Event Trigger" which executes a specific subroutine when a function key is pressed on keyboard.

Syntax ON KEY(*number%*) GOSUB *SubLabel*

Remarks "*number*" is a positive integer between 1 to 12 (1 to 9 for 711). It indicates the function key of the keyboard.

"*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Usage

```
ON KEY(1) GOSUB On_Shift
```

```
    ON KEY(2) GOSUB Off_Shift
    …
On_Shift:
    Mode$ = “IN”
    RETURN
Off_Shift:
    Mode$ = “OUT”
    RETURN
```

ON MINUTE_SHARP GOSUB…	**201**	**510**	**520**	**711**	**720**

Purpose To activate the “MINUTE_SHARP Event Trigger” which executes a specific subroutine when the system time is on the minute.

Syntax ON MINUTE_SHARP GOSUB *SubLabel*

Remarks “*SubLabel*“ is the name or line label of the subroutine to be called when the event is triggered.

Usage

```
    …
    ON MINUTE_SHARP GOSUB CheckTime
    …
CheckTime:
    CurrentTime$ = TIME$
    Hour% = VAL(MID$(CurrentTime$,3,2))
    IF Hour% = 30 THEN GOSUB HalfHourAlarm
    RETURN
    …
HalfHourAlarm:
    BEEP(800,30)
    WAIT(100)
    RETURN
```

ON NET GOSUB	**201**	**510**	**520**	~~711~~	~~720~~

Purpose To activate the “NET Event Trigger” which executes a specific subroutine when data is received from the RS-485 port.

Syntax ON NET GOSUB *SubLabel*

Remarks “*SubLabel*“ is the name or line label of the subroutine to be called when the event is triggered.

Usage

```
    ON NET GOSUB NetCommand
    …
NetCommand:
    OFF NET
    NetCmd$ = READ_NET$
    …
```

ON READER… GOSUB… 201 510 520 711 720

Purpose To activate the "READER Event Trigger" which executes a specific subroutine when data is received from the reader ports.

Syntax ON READER(*N%*) GOSUB *SubLabel*

Remarks "*N%*" can be 1 or 2 for terminals 201, 510, and 520.

"*N%*" is 1 for terminals 711and 720.

"*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Usage

```
ON READER(1) GOSUB BcrData_1
…
BcrData_1:
    OFF READER(1)
    BEEP(2000,5)
    Data$ = GET_READER_DATA$(1)
    …
```

ON TIMER… GOSUB… 201 510 520 711 720

Purpose To activate the "TIMER Event Trigger" which executes a specific subroutine when the system runs out the time duration that was specified by the user.

Syntax ON TIMER(*N%*, *duration*) GOSUB *SubLabel*

Remarks "*N%*" is an integer between 1 to 5 indicating the ordinal number of timer.

"*duration*" is specified in units of 10 ms.

"*SubLabel*" is the name or line label of the subroutine to be called when the event is triggered.

Up to five timers can be set in a BASIC program. Be sure the timer IDs are different. Otherwise the latter created timer will overwrite the former one.

Usage

```
ON TIMER(1,200) GOSUB ClearScreen     ' TIMER(1) = 2 sec
…
ClearScreen:
    OFF TIMER(1)
    CLS
    RETURN
```

4.5.2 Lock and Unlock

Event trapping could be nested. If the event triggers are activated in a BASIC program, it is possible that an event-driven subroutine can be interrupted by new coming events. Normally, the new event would be processed first. In some cases where we don't want the event-driven subroutine to be interrupted by other events, the LOCK and UNLOCK commands can be used to hold off new events.

LOCK	**201**	**510**	**520**	**711**	**720**

Purpose To hold all activated event triggers until they are released by the UNLOCK command.

Syntax LOCK

Remarks Command LOCK can prevent the nesting of event triggers. All the activated event triggers will be disabled until UNLOCK is called.

Usage

```
    ON READER(1) GOSUB BcrData_1
    ON READER(2) GOSUB BcrData_2
    …
BcrData_1:
    LOCK
    BEEP(2000,5)
    Data$ = GET_READER_DATA(1)$
    GOSUB AddNewData
    UNLOCK
    RETURN
    …
BcrData_2:
    BEEP(2000,5)
    Data$ = GET_READER_DATA(2)$
    GOSUB AddNewData
    RETURN
```

In this example, the BASIC program can trap the READER(1) and READER(2) events and reroute to the subroutines BcrData_1 and Bcr_Data_2 respectively. In BcrData_1, the command LOCK disables all the activated event triggers so that the subroutine BcrData_1 would not be interrupted by a new coming READER(1) and/ or READER(2) event. On the other hand, since LOCK is not called in BcrData_2, any new coming READER(1) and READER(2) event will interrupt the ongoing BcrData_2, and therefore may affect the expected results.

UNLOCK	**201**	**510**	**520**	**711**	**720**

Purpose To release all activated event triggers held by the LOCK command.

Syntax UNLOCK

Remarks

Usage Please refer to command LOCK.

4.6 System Commands

This section describes the system commands, such as the commands to change the CPU running speed, get the device ID, and/ or restart the system.

CHANGE_SPEED	~~201~~	~~510~~	~~520~~	**711**	**720**

Purpose To change the CPU running speed.

Syntax CHANGE_SPEED(*N%*)

Remarks "*N%*" is a numeric expression in the range of 1 to 5.

N% = 1, 1/16 speed.

N% = 2, 1/8 speed.

N% = 3, Quarter speed.

N% = 4, Half speed.

N% = 5, Full speed.

When the system is not heavy loaded, such as waiting for data input, it is suggested to change the CPU running speed to a lower speed to reduce the power consumption.

Usage CHANGE_SPEED(3)

DEVICE_ID$	~~201~~	~~510~~	~~520~~	**711**	**720**

Purpose To get the serial number of the terminal.

Syntax *A*$ = DEVICE_ID$

Remarks "*A$*" is a string variable to be assigned with the result.

A string of the terminal's serial number will be returned.

Usage PRINT "S/N : ", DEVICE_ID$

GET_STATION_ID	**201**	**510**	**520**	~~711~~	~~720~~

Purpose To get the station ID of the target terminal.

Syntax *A*% = GET_STATION_ID

Remarks "*A*%" is an integer variable to be assigned with the station ID of the terminal.

Usage StationID% = GET_STATION_ID

RESTART	**201**	**510**	**520**	**711**	**720**

Purpose To restart the system.

Syntax RESTART

Remarks The RESTART command terminates the execution of the BASIC program and restart it over again.

Usage HostCommand$ = READ_COM$(1)

…

IF HostCommand$ = “RESTART” THEN

RESTART

ELSE

…

SET_STATION_ID	**201**	**510**	**520**	~~711~~	~~720~~

Purpose To assign the station ID of the target terminal.

Syntax SET_STATION_ID(*N*%)

Remarks “*N*%” is the station ID assigned to the terminal.

For 201/510/520, it is used as the identification number for the RS-485 network. A valid value is in between 1 to 99.

Usage SET_STATION_ID(1)

VERSION	~~201~~	~~510~~	~~520~~	**711**	**720**

Purpose To write the version information to the system.

Syntax VERSION (Version$)

Remarks The user can use this command to write the version information, such as the program name and date to the system. The information written can be checked from the "Version" submenu of the “System Menu”.

Note that this command must be on the **first line** of the program or it will be ignored, and the string for the version information cannot exceed 15 characters.

Usage VERSION (“CipherBASIC 2.0”)

…

4.7 Reader Commands

The CipherLab terminals are able to read barcode/ magnetic-card data from the reader ports. This section describes the BASIC commands that are related to the reader ports of the terminals.

CODE_TYPE **201 510 520 711 720**

Purpose To get the type of symbology being decoded upon a successful scan.

Syntax *A%* = CODE_TYPE

Remarks "*A%*" is an integer variable to be assigned with the result.
The following list shows the possible values of the CODE_TYPE.

Symbology	Type	Symbology	Type
Code 39	65 (A)	EAN8 with Addon 2	78 (N)
Italy Pharma-code	66 (B)	EAN8 with Addon 5	79 (O)
CIP 39	67 (C)	EAN13 no Addon	80 (P)
Industrial 25	68 (D)	EAN13 with Addon 2	81 (Q)
Interleave 25	69 (E)	EAN13 with Addon 5	82 (R)
Matrix 25	70 (F)	MSI	83 (S)
Codabar (NW7)	71 (G)	Plessey	84 (T)
Code 93	72 (H)	Code ABC	85 (U)
Code128	73 (I)	ISO Track 1	97 (a)
UPCE no Addon	74 (J)	ISO Track 2	98 (b)
UPCE with Addon 2	75 (K)	ISO Track 1 and 2	99 (c)
UPCE with Addon 5	76 (L)	ISO Track 2 and 3	100 (d)
EAN8 no Addon	77 (M)	JIS II	101 (e)

Usage

```
…
CheckCodeType:
    IF CODE_TYPE = 65 THEN
            BcrType$ = "Code 39"
    ELSE IF CODE_TYPE = 66 THEN
            BcrType$ = "Italy Pharma-code"
    …
    END IF
    PRINT "Code Type : ", BcrType$
    RETURN
```

DISABLE READER **201 510 520 711 720**

Purpose To disable the reader ports of the terminal.

Syntax DISABLE READER (*N%*)

Remarks "*N%*" may be 1 or 2 for terminals 201, 510, and 520.
"*N%*" is 1 for terminals 711and 720.

Usage DISABLE READER (1)

ENABLE READER	201	510	520	711	720

Purpose To enable the reader ports of the terminal.

Syntax ENABLE READER (*N%*)

Remarks "*N%*" may be 1 or 2 for terminals 201, 510, and 520.

"*N%*" is 1 for terminals 711and 720.

The reader ports are disabled by default. With the 201/510/520 only the Master Setup Card can be read with the reader ports disabled. To enable barcode and magnetic card decoding function, the reader ports have to be enabled by the ENABLE READER command.

Usage

```
ENABLE READER (1)
ON READER (1) GOSUB Bcr_1
…
Bcr_1:
    Data$ = GET_READER_DATA$ (1)
    RETURN
```

GET_READER_DATA$	201	510	520	711	720

Purpose To get the data read from the specified reader port.

Syntax *A$* = GET_READER_DATA$ (*N%*)

Remarks "*A$*" is a string variable to be assigned with the result.

"*N%*" may be 1 or 2 for terminals 201, 510, and 520.

"*N%*" is 1 for terminals 711and 720.

Usually, the user uses ON READER GOSUB command to trap the event the data is transmitted to the terminal through the reader port, and then uses GET_READER_DATA$ command in a subroutine to get the reader data.

Usage

```
ON READER (1) GOSUB ReadData_1
…
ReadData_1:
    Data$ = GET_READER_DATA$ (1)
RETURN
```

GET_READER_SETTING	201	510	520	711	720

Purpose To get the value of the specified parameter of the barcode settings.

Syntax *A%* = GET_READER_SETTING(*N%*)

Remarks "*A%*" is a string variable to be assigned with the result.

"*N%*" is the number of the parameter, cf. READER_SETTING.

Usage

```
Setting1% = GET_READER_SETTING(1)
IF Setting1% = 1 THEN
    PRINT "Code 39 readability is enabled."
ELSE
    PRINT "Code 39 readability is disabled."
END IF
```

READER_SETTING	201	510	520	711	720

Purpose To set the value of the specified parameter of the barcode settings.

Syntax READER_SETTING (*N%, J%*)

Remarks "*N%*" is the number of the parameter.

"J%" is the value to be set to the parameter.

A set of parameters called barcode settings determines how the decoder will decode the barcode and magnetic card data. The initial values of the barcode settings are given by the Barcode Settings Window of the BASIC compiler; for details of the settings, please refer to Appendix D. The user can reset the values by using the READER_SETTING command in a BASIC program. The following table shows the details of these parameters.

No.	Value and Description
1	1 : Enable Code 39 0 : Disable Code 39
2	1 : Enable Italy Pharma-code 0 : Disable Italy Pharma-code
3	1 : Enable CIP 39 0 : Disable CIP 39
4	1 : Enable Industrial 25 0 : Disable Industrial 25
5	1 : Enable Interleave 25 0 : Disable Interleave 25
6	1 : Enable Matrix 25 0 : Disable Matrix 25
7	1 : Enable Codabar (NW7) 0 : Disable Codabar (NW7)
8	1 : Enable Code 93 0 : Disable Code 93
9	1 : Enable Code 128 0 : Disable Code 128
10	1 : Enable UPCE no Addon 0 : Disable UPCE no Addon
11	1 : Enable UPCE Addon 2 0 : Disable UPCE Addon 2
12	1 : Enable UPCE Addon 5 0 : Disable UPCE Addon 5
13	1 : Enable EAN8 no Addon 0 : Disable EAN8 no Addon
14	1 : Enable EAN8 Addon 2 0 : Disable EAN8 Addon 2
15	1 : Enable EAN8 Addon 5 0 : Disable EAN8 Addon 5
16	1 : Enable EAN13 no Addon 0 : Disable EAN13 no Addon
17	1 : Enable EAN13 Addon 2 0 : Disable EAN13 Addon 2
18	1 : Enable EAN13 Addon 5 0 : Disable EAN13 Addon 5
19	1 : Enable MSI 0 : Disable MSI
20	1 : Enable Plessey 0 : Disable Plessey
21	1 : Enable Code ABC 0 : Disable Code ABC

22	1 : Transmit Code 39 Start/Stop Character 0 : DO NOT Transmit Code 39 Start/Stop Character
23	1 : Verify Code 39 Check Character 0 : DO NOT Verify Code 39 Check Character
24	1 : Transmit Code 39 Check Character 0 : DO NOT Transmit Code 39 Check Character
25	1 : Full ASCII Code 39 0 : Standard Code 39
26	1 : Transmit Italy Pharmacode Check Character 0 : DO NOT Transmit Italy Pharmacode Check Character
27	1 : Transmit CIP39 Check Character 0 : DO NOT Transmit CIP39 Check Character
28	1 : Verify Interleave 25 Check Digit 0 : DO NOT Verify Interleave 25 Check Digit
29	1 : Transmit Interleave 25 Check Digit 0 : DO NOT Transmit Interleave 25 Check Digit
30	1 : Verify Industrial 25 Check Digit 0 : DO NOT Verify Industrial 25 Check Digit
31	1 : Transmit Industrial 25 Check Digit 0 : DO NOT Transmit Industrial 25 Check Digit
32	1 : Verify Matrix 25 Check Digit 0 : DO NOT Verify Matrix 25 Check Digit
33	1 : Transmit Matrix 25 Check Digit 0 : DO NOT Transmit Matrix 25 Check Digit
34	Select Interleave25 Start/Stop Pattern 0 : Use Industrial25 Start/Stop Pattern 1 : Use Interleave25 Start/Stop Pattern 2 : Use Matrix25 Start/Stop Pattern
35	Select Industrial25 Start/Stop Pattern 0 : Use Industrial25 Start/Stop Pattern 1 : Use Interleave25 Start/Stop Pattern 2 : Use Matrix25 Start/Stop Pattern
36	Select Industrial25 Start/Stop Pattern 0 : Use Industrial25 Start/Stop Pattern 1 : Use Interleave25 Start/Stop Pattern 2 : Use Matrix25 Start/Stop Pattern
37	Codabar Start/Stop Character 0 : abcd/abcd 1 : abcd/tn*e 2 : ABCD/ABCD 3 : ABCD/TN*E
38	1 : Transmit Codabar Start/Stop Character 0 : DO NOT Transmit Codabar Start/Stop Character
39	MSI Check Digit Verification 0 : Single Modulo 10 1 : Double Modulo 10 2 : Modulo 11 and Modulo 10
40	MSI Check Digit Transmission 0 : the last Check Digit is not transmitted 1 : both Check Digits are transmitted 2 : both Check Digits are not transmitted
41	1 : Transmit Plessey Check Characters 0 : DO NOT Transmit Plessey Check Characters
42	1 : Convert Standard Plessey to UK Plessey 0 : No Conversion
43	1 : Convert UPCE to UPCA 0 : No Conversion
44	1 : Convert UPCA to EAN13 0 : No Conversion
45	1 : Enable ISBN Conversion 0 : No Conversion

46	1 : Enable ISSN Conversion 0 : No Conversion
47	1 : Transmit UPCE Check Digit 0 : DO NOT Transmit UPCE Check Digit
48	1 : Transmit UPCA Check Digit 0 : DO NOT Transmit UPCA Check Digit
49	1 : Transmit EAN8 Check Digit 0 : DO NOT Transmit EAN8 Check Digit
50	1 : Transmit EAN13 Check Digit 0 : DO NOT Transmit EAN13 Check Digit
51	1 : Transmit UPCE System Number 0 : DO NOT Transmit UPCE System Number
52	1 : Transmit UPCA System Number 0 : DO NOT Transmit UPCA System Number
53	1 : Convert EAN8 to EAN13 0 : No Conversion
54	1 : Transmit Code ABC Concatenation Characters 0 : DO NOT Transmit Code ABC Concatenation Characters
55	1 : Enable Reversed Barcode 0 : Disable Reversed Barcode
56	0 : No Read Redundancy for Scanner Port 1 1 : One Time Read Redundancy for Scanner Port 1 2 : Two Times Read Redundancy for Scanner Port 1 3 : Three Times Read Redundancy for Scanner Port 1
57	0 : No Read Redundancy for Scanner Port 2 1 : One Time Read Redundancy for Scanner Port 2 2 : Two Times Read Redundancy for Scanner Port 2 3 : Three Times Read Redundancy for Scanner Port 2
58	1 : Industrial 25 Code Length Limitation in Max/Min Length Format 0 : Industrial 25 Code Length Limitation in Fix Length Format
59	Industrial 25 Max Code Length / Fixed Length 1
60	Industrial 25 Min Code Length / Fixed Length 2
61	1 : Interleave 25 Code Length Limitation in Max/Min Length Format 0 : Interleave 25 Code Length Limitation in Fix Length Format
62	Interleave 25 Max Code Length / Fixed Length 1
63	Interleave 25 Min Code Length / Fixed Length 2
64	1 : Matrix 25 Code Length Limitation in Max/Min Length Format 0 : Matrix 25 Code Length Limitation in Fix Length Format
65	Matrix 25 Max Code Length / Fixed Length 1
66	Matrix 25 Min Code Length / Fixed Length 2
67	1 : MSI Code Length Limitation in Max/Min Length Format 0 : MSI Code Length Limitation in Fix Length Format
68	MSI 25 Max Code Length / Fixed Length 1
69	MSI Min Code Length / Fixed Length 2
70	Scan Mode for Scanner Port 1 0 : Auto Off Mode 1 : Continuous Mode 2 : Auto Power Off Mode 3 : Alternate Mode 4 : Momentary Mode 5 : Repeat Mode 6 : Laser Mode 7 : Test Mode 8 : Aiming Mode
71	Scan Mode for Scanner Port 2 0 : Auto Off Mode 1 : Continuous Mode 2 : Auto Power Off Mode 3 : Alternate Mode 4 : Momentary Mode 5 : Repeat Mode

	6 : Laser Mode 7 : Test Mode 8 : Aiming Mode
72	Scanner Time-out Duration in seconds for Scanner Port 1
73	Scanner Time-out Duration in seconds for Scanner Port 2

Usage READER_SETTING(1,1)

' Code 39 readability is enabled.

4.8 Keyboard Wedge Commands

Terminals 201, 711 and 720 are able to send data to the host through the keyboard wedge interface by using command SEND_WEDGE. The SEND_WEDGE function is governed by a set of parameters called Wedge Settings. Command SET_WEDGE is used to configure these parameters.

4.8.1 Definition of the *WedgeSetting* array

WedgeSetting$ is a 3-element character array passed to SET_WEDGE to describe the characteristics of the keyboard wedge interface. In a BASIC program, WedgeSetting$ can be defined as,

WedgeSetting$ = Wedge_1$ + Wedge_2$ + Wedge_3$.

The functions of the parameters Wedge_1$, Wedge_2$, and Wedge_3$ are described in the following subsections.

Parameter	Bit	Description
Wedge_1$	7 - 0	KBD / Terminal Type
Wedge_2$	7	1 : enable capital lock auto-detection 0 : disable capital lock auto-detection
Wedge_2$	6	1 : capital lock on 0 : capital lock off
Wedge_2$	5	1 : ignore alphabets case 0 : alphabets are case sensitive
Wedge_2$	4 - 3	00 : normal 10 : digits are at lower position 11 : digits are at upper position
Wedge_2$	2-1	00 : normal 10 : capital lock keyboard 11 : shift lock keyboard
Wedge_2$	0	1 : use numeric key pad to transmit digits 0 : use alpha-numeric key to transmit digits
Wedge_3$	7 - 0	inter-character delay

4.8.2 KBD / Terminal Type

The first element determines which type of keyboard wedge is applied. The possible value is listed as follows.

Setting Value	Terminal Type	Setting Value	Terminal Type
0	Null (Data not Transmitted)	21	PS55 002-81, 003-81
1	PCAT (US)	22	PS55 002-2, 003-2
2	PCAT (FR)	23	PS55 002-82, 003-82
3	PCAT (GR)	24	PS55 002-3, 003-3
4	PCAT (IT)	25	PS55 002-8A, 003-8A
5	PCAT (SV)	26	IBM 3477 TYPE 4
6	PCAT (NO)	27	PS2-30
7	PCAT (UK)	28	Memorex Telex 122 Keys
8	PCAT (BE)	29	PCXT
9	PCAT (SP)	30	IBM 5550
10	PCAT (PO)		
11	PS55 A01-1		
12	PS55 A01-2	33	DEC VT220,320,420
13	PS55 A01-3		
14	PS55 001-1		

15	PS55 001-81		
16	PS55 001-2		
17	PS55 001-82		
18	PS55 001-3		
19	PS55 001-8A		
20	PS55 002-1, 003-1		

For example, if the terminal type is PCAT (US), then the first element of the WedgeSetting$ can be defined as,

Wedge_1$ = CHR$(1)

4.8.3 Capital Lock Auto-Detection

If “Capital Lock Auto-Detection” is enabled, the SEND_WEDGE function can automatically detect the capital lock status of keyboard when the keyboard type selected is PCAT (all available languages), PS2-30, PS55, or Memorex Telex. If this is the case, the SEND_WEDGE function will ignore the capital lock status setting and perform auto-detection when transmitting data. If “Capital Lock Auto-Detection” is disabled, the SEND_WEDGE function will transmit alphabets according to the setting of the capital lock status.

If the keyboard type selected is neither PCAT, PS2-30, PS55, nor Memorex Telex, the SEND_WEDGE function will transmit the alphabets according to setting of the capital lock status even though the auto-detection setting is enabled.

To enable “Capital Lock Auto-Detection”, add 128 to the value of the second element of WedgeSetting$ (Wedge_2$).

4.8.4 Capital Lock Status Setting

To send alphabets with correct case (upper or lower case), the SEND_WEDGE function must know the capital lock status of keyboard when transmitting data. Incorrect capital lock setting will result in different letter case (‘A’ becomes ‘a’, and ‘a’ becomes ‘A’).

To set “Capital Lock ON”, add 64 to the value of the second element of WedgeSetting$ (Wedge_2$).

4.8.5 Alphabets Case

The setting of this bit affects the way the SEND_WEDGE function transmits alphabets. The SEND_WEDGE function can transmit alphabets according to their original case (case sensitive) or just ignore it. If ignoring case is selected, the SEND_WEDGE function will always transmit alphabets without adding shift key.

To set “Ignore Alphabets Case”, add 32 to the value of the second element of WedgeSetting$ (Wedge_2$).

4.8.6 Digits Position

This setting can force the SEND_WEDGE function to treat the position of the digit keys on the keyboard differently. If this setting is set to upper, the SEND_WEDGE function will add shift key when transmitting digits. This setting will be effective only when the keyboard type selected is PCAT (all available language), PS2-30, PS55, or Memorex Telex. Also if the user chooses to send digits using numeric keypad, then this setting is meaningless.

To set “Lower Position”, add 16 to the value of the second element of WedgeSetting$ (Wedge_2$). To set “Upper Position”, add 24 to the value of the second element of WedgeSetting$ (Wedge_2$). Do not set this setting unless the user is sure about the selection.

4.8.7 Shift / Capital Lock Keyboard

This setting can force the SEND_WEDGE function to treat the keyboard type to be a shift lock keyboard or a capital lock keyboard. This setting will be effective only when the keyboard type selected is PCAT (all available language), PS2-30, PS55, or Memorex Telex.

To set “Capital Lock”, add 4 to the value of the second element of WedgeSetting$ (Wedge_2$). To set “Shift Lock”, add 6 to the value of the second element of WedgeSetting$ (Wedge_2$). Do not set this setting unless the user is sure about the selection.

4.8.8 Digit Transmission

This setting instructs the SEND_WEDGE function which group of keys are used to transmit digits, whether to use the digit keys on top of the alphabet keys or use the digit keys on the numeric keypad.

To set “Use Numeric Keypad to Transmit Digits”, add 2 to the value of the second element of WedgeSetting$ (Wedge_2$).

4.8.9 Inter-Character Delay

A 0 to 255 ms inter-character delay can be added before transmitting each character. This is used to provide some response time for PC to process keyboard input.

For example, to set the inter-character delay to be 10 ms, the third element of the WedgeSetting$ can be defined as,

```
Wedge_3$ = CHR$(10)
```

4.8.10 Composition of Output String

The keyboard wedge character map is shown below. Each character in the output string is translated by this table when SEND_WEDGE function transmits data.

	00	10	20	30	40	50	60	70	80
0		F2	SP	0	@	P	`	p	⓪
1	INS	F3	!	1	A	Q	A	q	①
2	DLT	F4	"	2	B	R	B	r	②
3	Home	F5	#	3	C	S	C	s	③
4	End	F6	$	4	D	T	D	t	④
5	Up	F7	%	5	E	U	E	u	⑤
6	Down	F8	&	6	F	V	F	v	⑥
7	Left	F9	'	7	G	W	G	w	⑦
8	BS	F10	(	8	H	X	H	x	⑧
9	HT	F11	)	9	I	Y	I	y	⑨
A	LF	F12	*	:	J	Z	J	z	
B	Right	ESC	+	;	K	[	K	{	
C	PgUp	Exec	,	<	L	\	L	|	
D	CR	CR*	-	=	M	]	M	}	
E	PgDn		.	>	N	^	N	~	
F	F1		/	?	O	_	O	Dly	

Dly :
Delay 100 ms

⓪…⑨ :
Digits of Numeric Key Pad

CR* : Enter key on the numeric key pad

The SEND_WEDGE function can not only transmit simple characters as above, but also provide a way to transmit combination key status, or even direct scan codes. This is done by inserting some special command codes in the output string. A command code is a character whose value is between 0xC0 and 0xFF.

0xC0 : Indicates that the next character is to be treated as scan code. Transmit it as it is, no translation required.

0xC0 | 0x01 : Send next character with Shift key.

0xC0 | 0x02 : Send next character with left Ctrl key.

0xC0 | 0x04 : Send next character with left Alt key.

0xC0 | 0x08 : Send next character with right Ctrl key.

0xC0 | 0x10 : Send next character with right Alt key.

0xC0 | 0x20 : Clear all combination status key after sending the next character.

For example, to send **[A] [Ctrl-Insert] [5] [scan code 0x29] [Tab] [2] [Shift-Ctrl-A] [B] [Alt-1] [Alt-2-Break] [Alt-1] [Alt-3]**, the following characters are inserted into the string supplied to the SEND_WEDGE function.

0x41, 0xC2, 0x01, 0x35, 0xC0, 0x29, 0x09, 0x32, 0xC3, 0x41, 0x42, 0xC4, 0x31 0xE4, 0x32, 0xC4, 0x31, 0xC4, 0x33

Please note that, the scan code 0x29 is actually a space for PCAT, Alt-12 is a form feed character, and Alt-13 is an Enter. The **break** after Alt-12 is necessary, if omitted the characters will be treated as Alt-1213 instead of Alt-12 and Alt-13.

The following instructions can be called in the BASIC program to send the above string through the keyboard wedge intergace.

```
…
Data_1$ = CHR$(65) + CHR$(194) + CHR$(1) + CHR$(53) + CHR$(192) + CHR$(41)
Data_2$ = CHR$(9) + CHR$(50) + CHR$(195) + CHR$(65) + CHR$(66)
Data_3$ = CHR$(196) + CHR$(49) + CHR$(228) + CHR$(50) + CHR$(196) + CHR$(49)
Data_4$ = CHR$(196) + CHR$(51)
DataString$ = Data_1$ + Data_2$ + Data_3$ + Data_4$
SEND_WEDGE(DataString$)
…
```

4.8.11 Special Note on DEC VT220/320/420 (for 201 only)

Because the DEC VT220/320/420 is using RS-232 to communicate between DEC and its keyboard, the following two lines must be included into the program, if this keyboard type is selected.

```
SET_COM (1, 5, 1, 2, 1)
SET_COM (2, 5, 1, 2, 1)
OPEN_COM (1)
OPEN_COM (2)
```

These lines must be executed immediately after the **WedgeSetting** is initialized. Please note that the 7xx terminal cannot send wedge data to DEC VT220/320/420.

SEND_WEDGE	201	~~510~~	~~520~~	711	720

Purpose To send data to the host via keyboard wedge interface.

Syntax SEND_WEDGE (*DataString$*)

Remarks "*DataString$*" is the data string to be sent through the keyboard wedge interface.

Usage

```
…
DataString$ = CHR$(9) + "TESTING" + CHR$(9)
' [Tab] + "TESTING" + [Tab]
SEND_WEDGE(DataString$)
…
```

SET_WEDGE	201	~~510~~	~~520~~	711	720

Purpose To configure the keyboard wedge interface.

Syntax SET_WEDGE$ (*WedgeSetting$*)

Remarks "*WedgeSetting$*" is a 3-element character array describing the characteristics of the keyboard wedge interface.

Usage

```
…
Wedge_1$ = CHR$(1)              ' terminal type: PCAT(US)
Wedge_2$ = CHR$(1)
' auto-detection disabled, capital lock off, case sensitive
' use numeric key pad to transmit digits
Wedge_3$ = CHR$(5)              ' inter-char-delay: 5 ms
WedgeSetting$ = Wedge_1$ + Wedge_2$ + Wedge_3$
SET_WEDGE(WedgeSetting$)
…
```

4.9 Buzzer

This section describes the beeper-related commands.

BEEP	201	510	520	711	720

Purpose To make beeper sounds according to the specified beeper sequence.

Syntax BEEP (*freq%, duration%* {, *freq%, duration%*}

Remarks *freq%* must be an integer that indicates the value of beep frequence (Hz). Suitable frequency for the beeper ranges from 1 kHz to 6 kHz.

duration% must be an integer that indicates the value of beep duration. Beep duration is specified in units of 10 ms.

Up to eight frequency-duration pairs can be assigned in a BEEP command. If the value of the frequency is 0, the beeper will not beep during the time duration.

Usage

```
    ON READER(1) GOSUB BcrData_1
    …
BcrData_1:
    BEEP(2000, 10, 0, 10, 2000, 10)
    …
    RETURN
```

KEY_CLICK	~~201~~	~~510~~	520	711	720

Purpose To enable/disable the key click sound.

Syntax KEY_CLICK(*status%*)

Remarks "*status%*" may be 1 or 0.

0: disable the key click sound.

1: enable the key click sound.

The key click is enabled by default.

Usage

```
KEY_CLICK(0)    REM disable the key click
```

STOP BEEP	201	510	520	711	720

Purpose To terminate the beeps.

Syntax STOP BEEP

Remarks The STOP BEEP statement terminates the beep immediately if there is a beeper sequence in progress.

Usage

```
    BEEP(2000, 0)
    ON KEY (1) GOSUB StopBeep
    PRINT "Press F1 to stop the beeper."
    …..
StopBeep:
    STOP BEEP
    RETURN
```

VOL	201	510	520	711	720

Purpose To set the volume of the beeper.

Syntax VOL (*level%*)

Remarks *level%,* an integer in the range of 0 to 3, indicates the volume level (from Min to Max) of the beeper.

Usage

```
VOL (0)          ' The beeper volume is set to minimum level.
```

4.10 Calendar and Timer Commands

This section describes the calendar- and timer-related commands. The system date and time are kept by the calendar chip, and they can be retrieved from or set to the calendar chip by the DATE$ and TIME$ functions. A backup rechargeable battery keeps the calendar chip running even when power is turned off.

Note that the system time variable TIMER is maintained by CPU timers and has nothing to do with this calendar chip. Accuracy of this time variable depends on the CPU clock and is not suitable for precise time manipulation. Also, it is reset to 0 upon power up (cold start).

Up to five timers can be set by the ON TIMER… GOSUB… command for the "TIMER Event Trigger".

DATE$	**201**	**510**	**520**	**711**	**720**

Purpose To set or to get the current date.

Syntax DATE$ = *X$* - To set the current date.
*Y$* = DATE$ - To get the current date.

Remarks "*X$*" is in the form of "yyyymmdd".
"*Y$*" is a valid string variable.
The BASIC Compiler and its Run-time Engines do not check the format and contents of the string to be assigned to DATE$. The user has the responsibility to check the format and contents.

Usage
```
DATE$ = "20000103"
Today$ = DATE$
PRINT Today$        ' Today$ = "20000103"
…
```

DAY_OF_WEEK	~~**201**~~	~~**510**~~	**520**	**711**	**720**

Purpose To get the day of the week.

Syntax *A*% = DAY_OF_WEEK

Remarks "*A%*" is an integer variable to be assigned with the result.
A value of 1 to 7 represents Monday to Sunday respectively.

Usage
```
    ON DAY_OF_WEEK GOSUB 100, 200, 300, 400, 500, 600, 700
    …
100
    PRINT "Today is Monday."
    RETURN
200
    PRINT "Today is Tuesday."
    RETURN
300
    PRINT "Today is Wednesday."
    RETURN
    …
```

HIDE_CALENDAR	201	510	520	~~711~~	~~720~~

Purpose To hide the calendar, which is displayed on the LCD by the SHOW_CALENDAR command.

Syntax HIDE_CALENDAR

Remarks This is the opposite of the SHOW_CALENDAR command.

Usage

```
SHOW_CALENDAR(1,1,1)
ON READER(1) GOSUB BcrData_1
…
Bcr_Data_1:
HIDE_CALENDAR
BEEP(2000,5)
Data$ = GET_READER_DATA$(1)
PRINT Data$
…
```

SHOW_CALENDAR	201	510	520	~~711~~	~~720~~

Purpose To show the calendar on the LCD.

Syntax SHOW_CALENDAR (*mode%*, *y_pos%*, *x_pos%*)

Remarks "*mode%*" may be 1, 2, or 3. It defines the format of the calendar to be shown.

1: yyyy/mm/dd hh:mm:ss

2: mm/dd/yyyy hh:mm:ss

3: dd/mm/yyyy hh:mm:ss

"*y_pos%*" is an integer which indicates the row number where the calendar to be displayed.

"*x_pos%*" is an integer which indicates the column number where the calendar to be displayed.

Usage

```
SHOW_CALENDAR(1,1,1)
ON READER(1) GOSUB BcrData_1
…
Bcr_Data_1:
HIDE_CALENDAR
BEEP(2000,5)
Data$ = GET_READER_DATA$(1)
PRINT Data$
…
```

TIME$	201	510	520	711	720

Purpose To set or get the current time.

Syntax TIME$ = *X$* - To set the current time.

*Y$* = TIME$ - To get the current time.

Remarks "*X$*" is in the form of "hhmmss".

"*Y$*" is a valid string variable.

The BASIC Compiler and its Run-time Engines do not check the format and contents of the string to be assigned to TIME$. The user has the responsibility to check the format and contents.

Usage

```
CurrentTime$ = TIME$   'assign the current time to CurrentTime$
TIME$ = "112500"       ' set the system time to 11:25:00
```

TIMER	201	510	520	711	720

Purpose To return the number of seconds since the terminal is powered on.

Syntax A& = TIMER

Remarks "*A&*" is a long integer variable to be assigned with the result.

TIMER is a read only function. The system timer cannot be set by using the TIMER function.

Usage

```
    StartTime& = TIMER
    …
Loop:
    IF EndTime& <> TIMER THEN
        EndTime& = TIMER
        TimerElapsed& = EndTime& - StartTime&
        CLS
        PRINT TimerElapsed&
        IF TimerElapsed& > 100 THEN GOTO NextStep
    END IF
    GOTO Loop
NextStep:
    …
```

WAIT	201	510	520	711	720

Purpose To put the system on hold for the specified time duration. During this period, the system will be running in a special low-power-consumption mode.

Syntax WAIT(*duration%*)

Remarks "*duration%*" must be a positive integer that indicates the time duration to hold. This argument is specified in units of 5 ms.

When the application is waiting for events in a loop, by calling this function can dramatically reduce the power consumption.

Usage

```
PRINT "CipherLab BASIC"
WAIT(200)     ' The system is on hold for 1 second.
```

4.11 Digital I/O Commands

Terminals 201, 510, and 520 provide digital I/Os for controlling and monitoring external devices. This section describes the commands concerning the digital I/Os.

DIGOUT **201 510 520** ~~711~~ ~~720~~

Purpose To set the digital ouput.

Syntax DIGOUT(*number%*,*mode%*,*duration%*)

Remarks "*number*" is a positive integer that indicates the digital output port. For 201 and 510, it can be 1 or 2. For 520, it can be 1 to 4.

"*mode%*" may be 1, 2, or 3. It indicates the digital output mode. The meaning for each mode are listed below.

Mode	Interpretation
1	Turn on the digital output for the specific duration and then go back off.
2	Turn off the digital output for the specific duration and then go back on.
3	Flash the I/O with a specific period indefinitely. The flashing period equals 2 X *duration*.

"*duration%*" is an integer number which specifies the time duration in units of 10 ms. A value of 0 in this argument will keep the I/O state in the specific state indefinitely.

Usage

```
    ON READER(1) GOSUB BcrData_1
    …
BcrData_1:
    BEEP(2000,5)
    Data$ = GET_READER_DATA$(1)
    IF FIND_RECORD(1,1,Data$) THEN
        DIGOUT(1,1,100)
    ELSE
        PRINT "Invalid Card ID."
    END IF
    …
```

GET_DIGIN **201 510 520** ~~711~~ ~~720~~

Purpose To read the digital input state.

Syntax *A*% = GET_DIGIN(*number%*)

Remarks "*number*" is a positive integer indicates the digital input port. For 201 and 510, it can be 1 or 2. For 520, it can be 1 to 4.

The current state of the specified digital input port, 0 (Low) or 1 (High), will be returned.

Usage IF GET_DIGIN(1) THEN PRINT "Digital Input (1) is High."

4.12 LED Command

Terminals 201, 510, 520, 711 and 720 provide different numbers of LEDs, which can be used to show the system status. This section describes the command concerining the LEDs.

LED	201	510	520	711	720

Purpose To set the LEDs.

Syntax LED(*number%*,*mode%*,*duration%*)

Remarks "*number*" is a positive integer indicates the LED number. Number of LEDs provided depends on the type of the terminal.

201

Number	Name
1	GOOD READ (Green)
2	GOOD READ (Green)
4	POWER (Red)

510

Number	Name
3	F4
4	READY (Yellow)
5	GOOD READ (Green)
6	F3
7	F2
8	BAT. LOW (Red)
9	SHIFT
10	F1

520

Number	Name
1	F1
2	F2
3	F3
4	F4
5	F5
6	F6
7	F7
8	F8
9	ALPHA
10	POWER (Red)
11	GOOD READ (Green)

711/720

Number	Name
1	Red LED
2	Green LED

"*mode%*" may be 1, 2, or 3. It indicates the digital output mode. The values of the mode and their interpretation are listed below.

Mode	Interpretation
1	Turn on the LED for the specific duration and then go back off.
2	Turn off the LED for the specific duration and then go back on.
3	Flash the LED with a specific period indefinitely. The flashing period equals 2X*duration*.

"*duration%*" is an integer number which specifies the time duration in units of 10 ms. A value of 0 in this argument will keep the LED in the specific state indefinitely.

Usage

```
ON READER(1) GOSUB BcrData_1
…
BcrData_1:
BEEP(2000,5)
LED(11, 1, 5)     ' GOOD READ LED for 520
Data$ = GET_READER_DATA$(1)
…
```

4.13 Keyboard Commands

Terminals 510, 520, 711 and 720 provide the built-in keypads for data input. The user can also use external AT keyboards for 201, 510 and 520 terminals. This section describes the keyboard-related commands.

ALPHA_LOCK	~~201~~	~~510~~	**520**	**711**	**720**

Purpose Forces the alpha-input status to be on or off.

Syntax ALPHA_LOCK (*status%*)

Remarks "*status%*" may be 1 or 0.
0: unlock, meaning alpha-input status is off.
1: lock, meaning alpha-input status is on.

Usage ALPHA_LOCK (1)
…

CLR_KBD	**201**	**510**	**520**	**711**	**720**

Purpose To clear the keyboard buffer.

Syntax CLR_KBD

Remarks By calling this function, the data that is still queued in the keyboard buffer will be cleared.

Usage CLR_KBD
ON KEY(1) GOSUB KeyData_1
…

CAP_LOCK	**201**	**510**	**520**	~~711~~	~~720~~

Purpose To set the Caps Lock status for the external AT keyboard.

Syntax CAP_LOCK (*status%*)

Remarks "*status%*" may be 1 or 0.
0: Caps Lock ON.
1: Caps Lock OFF.

Usage CAP_LOCK (1)

DISABLE KBD	**201**	**510**	**520**	**711**	**720**

Purpose To disable the built-in keypad and/ or the external AT keyboard.

Syntax DISABLE KBD

Remarks Once executed, the keyboard-input functionality will be disabled.

Usage ENABLE KBD
…
INPUT String1$
BEEP(2000,5)
DISABLE KBD
PRINT String1$
…

DISABLE PSEUDO_KBD	**201**	**510**	**520**	~~711~~	~~720~~

Purpose To disable the pseudo keyboard as well as the real keyboard.

Syntax DISABLE PSEUDO_KBD

Remarks Pseudo keyboard is not a real keyboard, but a special set of barcodes. Each barcode represents a key on the real keyboard. When pseudo keyboard function is enabled, users can make a keystroke by scanning the corresponding barcode. By reading the keyboard, keystrokes are input to the terminal as if they come from a real keyboard. A set of pseudo keyboards can be found in Appendix E.

Usage ENABLE PSEUDO_KBD
…
INPUT String1$
BEEP(2000,5)
DISABLE PSEUDO_KBD
PRINT String1$
…

ENABLE KBD	**201**	**510**	**520**	**711**	**720**

Purpose To enable the built-in keypad and/ or the external AT keyboard.

Syntax ENABLE KBD

Remarks The keypad functionality is enabled by default. But once disabled by the "DISABLE KBD" command, this command can be used to restore the keypad functionality.

Usage ENABLE KBD
…
INPUT String1$
BEEP(2000,5)
DISABLE KBD
PRINT String1$
…

ENABLE PSEUDO_KBD	**201**	**510**	**520**	~~711~~	~~720~~

Purpose To enable the pseudo keyboard as well as the real keyboard.

Syntax ENABLE PSEUDO_KBD

Remarks Pseudo keyboard is not a real keyboard, but a special set of barcodes. Each barcode represents a key on the real keyboard. When pseudo

keyboard function is enabled, the user can enter a keystroke by scanning the corresponding barcode. By reading the barcode, keystrokes are input to the terminal as if they come from a real keyboard. A set of pseudo keyboards can be found in Appendix E.

Usage

```
ENABLE PSEUDO_KBD
…
INPUT String1$
BEEP(2000,5)
DISABLE PSEUDO_KBD
PRINT String1$
…
```

GET_ALPHA_LOCK ~~201~~ ~~510~~ **520 711 720**

Purpose To get alpha-lock status.

Syntax *A%* = GET_ALPHA_LOCK

Remarks "*A%*" is an integer variable to be assigned with the alpha-lock status.
The current alpha-lock status, 0 (unlocked) or 1 (locked) will be returned.

Usage

```
Alpha_lock% = GET_ ALPHA_LOCK
```

GET_SHIFT_LOCK ~~201~~ ~~510~~ ~~520~~ ~~711~~ **720**

Purpose To get shift-lock status.

Syntax *A%* = GET_SHIFT_LOCK

Remarks "*A%*" is an integer variable to be assigned with the shift-lock status.
The current shift-lock status, 0 (unlocked) or 1 (locked) will be returned.

Usage

```
Shift_lock% = GET_SHIFT_LOCK
```

INKEY$ **201 510 520 711 720**

Purpose To read a character from the built-in keypad and/ or the external AT keyboard.

Syntax *X$* = INKEY$

Remarks "*X$*" is a string variable that is assigned with the character read.

Usage

```
    …
    PRINT "Initialize System (Y/N)?"
Loop:
    KeyData$ = INKEY$
    IF KeyData$ = "" THEN
        GOTO Loop
    ELSE IF KeyData$ = "Y" THEN
        GOTO Initialize
    …
```

INPUT	201	510	520	711	720

Purpose To take input from keypad and store the data in a variable.

Syntax INPUT *variable*

Remarks "*variable*" is a numeric or string variable that will receive the input data. The data entered must match the data type of the variable.

Usage

```
INPUT String1$      ' Input a string variable
PRINT String$
INPUT Number%       ' Input a numeric variable
PRINT Number%
```

INPUT_MODE	201	510	520	711	720

Purpose To set the display mode of the input data.

Syntax INPUT_MODE(*mode%*)

Remarks "*mode%*" may be 1, 2, or 3.

1: The input characters will be displayed on the LCD.

2: "*" will be displayed instead of the characters typed in.

3: Nothing will be displayed on the LCD.

Usage

```
LOCATE 1,1
INPUT_MODE(1)
INPUT Login$
LOCATE 2,1
INPUT_MODE(2)
INPUT Password$
```

NUM_LOCK	201	510	520	~~711~~	~~720~~

Purpose To set the Num Lock status of the external AT keyboard.

Syntax NUM_LOCK(*status%*)

Remarks "*status%*" may be 1 or 0.

0: Num Lock is ON.

1: Num Lock is OFF.

Usage NUM_LOCK(1)

SHIFT_LOCK	~~201~~	~~510~~	~~520~~	~~711~~	720

Purpose Forces the shift-lock status to be on or off

Syntax SHIFT_LOCK(*status%*)

Remarks "*status%*" may be 1 or 0.

0: unlock, meaning the shift status is off.

1: lock, meaning the shift status is on.

Usage

```
SHIFT_LOCK(1)
…
```

4.14 LCD Commands

This section describes the commands related to the LCD display.

4.14.1 Graphic Display

A 20 characters X 2 lines LCD display is supported by terminals 201 and 510. Since the LCD display is not a graphic display, the 201 and 510 cannot show graphic image on the LCD and also the font size is fixed to 6X8 (pixels). A 240X64 (pixels) graphic display is used on 520, and 128X64 (pixels) graphic display is used on 711 and 720. These are able to show graphics and select different font sizes.

A coordinates system is used in the cursor movement statements to determine the position of the cursor. The coordinates of the top left position are assigned with (1,1). For 201 and 510, the coordinates of the bottom right postion are assigned with (20,1). However, for 520, 711 and 720, the coordinates of the bottom right position depend on the font size.

4.14.2 Special Font Files

Besides the standard font, terminals 520, 711, and 720 can display special characters such as the Oriental language characters providing that those font files have been downloaded to the terminals. CipherLab provides users four special font files to display Japanese, Simplified Chinese, and Traditional Chinese characters.

Font files:

- Font-jp.shx: Japanese Characters Font File (16X16)
- Font-sc.shx: Simplified Chinese Characters Font File (16X16)
- Font-tc.shx: Traditional Chinese Characters Font File (16X16)
- Font-sd.shx: Simplified Chinese Characters Font File (12X12)

BACKLIT	**201**	**510**	**520**	**711**	**720**

Purpose To set the background intensity of the LCD.

Syntax BACKLIT(*status%*)

Remarks "*status%*" may be 0, 1, 2, or 3. It indicates the intensity of the LCD backlit from darkest to brightest.

Usage BACKLIT(3)

CLR_RECT	~~201~~	~~510~~	~~520~~	**711**	**720**

Purpose To clear a rectangle area.

Syntax CLR_RECT (*x%, y%, size_x%, size_y%*)

Remarks "*x%*" is the x coordinate of the upper left point of the rectangle area.
"*y%*" is the y coordinate of the upper left point of the rectangle area.
"*size_x%*" is the width of the rectangle area in pixels.
"*size_y%*" is the height of the rectangle area in pixels.

Usage CLR_RECT(1, 1, 20, 20)
…

CLS	201	510	520	711	720

Purpose To clear the LCD display.

Syntax CLS

Remarks After executing this command, whatever is showing on the LCD display will be erased and the cursor will be moved to (1,1).

Usage

```
ON TIMER(1,200) GOSUB ClearScreen   ' TIMER(1) = 2 sec
…
ClearScreen:
  OFF TIMER(1)
  CLS
  RETURN
```

CURSOR	201	510	520	711	720

Purpose To turn on / off the cursor of the LCD display.

Syntax CURSOR (*status%*)

Remarks "*status%*" may be 0, or 1.

0: cursor is OFF.

1: cursor is ON.

Usage CURSOR(0)

CURSOR_X	201	510	520	711	720

Purpose To get the current x-coordinate of the cursor.

Syntax *X%* = CURSOR_X

Remarks "*X%*" is an integer variable to be assigned with the column position of the cursor.

Usage

```
ON READER(1) GOSUB BcrData_1
…
BcrData_1:
  BEEP(2000,5)
  Data$ = GET_READER_DATA$(1)
  Pre_X% = CURSOR_X
  Pre_Y% = CURSOR_Y
  LOCATE 8, 1
  PRINT Data$
  LOCATE Pre_Y%, Pre_X%
  RETURN
```

CURSOR_Y	201	510	520	711	720

Purpose To get the current y-coordinate of the cursor.

Syntax *Y%* = CURSOR_Y

Remarks "*Y%*" is an integer variable to be assigned with the row position of the cursor.

Usage

```
    ON READER(1) GOSUB BcrData_1
    …
BcrData_1:
    BEEP(2000,5)
    Data$ = GET_READER_DATA$(1)
    Pre_X% = CURSOR_X
    Pre_Y% = CURSOR_Y
    LOCATE 8, 1
    PRINT Data$
    LOCATE Pre_Y%, Pre_X%
    RETURN
```

FILL_RECT	~~201~~	~~510~~	~~520~~	711	720

Purpose To fill a rectangular area.

Syntax FILL_RECT (*x%, y%, size_x%, size_y%*)

Remarks "*x%*" is the x coordinate of the upper left point of the rectangular area.
"*y%*" is the y coordinate of the upper left point of the rectangular area.
"*size_x%*" is the width of the rectangular area in pixels.
"*size_y%*" is the height of the rectangular area in pixels.

Usage

```
FILL_RECT (1, 1, 20, 20)
…
```

LOCATE	201	510	520	711	720

Purpose To move the cursor to the specified position on the LCD.

Syntax LOCATE *row%, col%*

Remarks "*row%*" is the new row position of the cursor.
"*col%*" is the new column position of the cursor.

Usage

```
LOCATE 1,1      ' Move cursor to the top left of the LCD.
```

PRINT	201	510	520	711	720

Purpose To display data on the LCD display.

Syntax PRINT *expression* [{,|;[*expression*]}]

Remarks "*expression*" may be numeric or string expression.

The position of each printed item is determined by the punctuation used to separate the items in the list. In the list of expression, a comma causes the next character to be printed after the last character with a blank space. A semicolon causes the next character to be printed immediately after the last value. **I**f the list of expressions terminates without a comma or a semicolon, a carriage return is printed at the end of the line.

Usage

```
LOCATE 1,1
PRINT STRING$(20, " ")   ' clear whole line
LOCATE 1,1
A = 5
PRINT A, "square is "; A*A
```

SELECT_FONT	~~201~~	~~510~~	**520**	**711**	**720**

Purpose To select different font size for the LCD display.

Syntax SELECT_FONT(*font%*)

Remarks "*font%*" may be 1, 2, or 3.

1: font size: 6X8

2: font size: 8X16

3: font size: 16X24 (for 520 only)

Note that the 16X24 font size is not available for terminals 711 and 720.

Usage SELECT_FONT(2)

SET_VIDEO_MODE	~~201~~	~~510~~	**520**	**711**	**720**

Purpose To set the display mode for the LCD.

Syntax SET_VIDEO_MODE(*mode%*)

Remarks "*mode%*" may be 0, or 1.

0: normal mode

1: reverse mode

Usage

```
SET_VIDEO_MODE(1)
PRINT "CipherLab 520"   ' This string will be printed in reverse mode.
```

SHOW_IMAGE	~~201~~	~~510~~	**520**	**711**	**720**

Purpose To show a bitmap on the LCD display.

Syntax SHOW_IMAGE (*x%, y%, size_x%, size_y%, image$*)

Remarks "*x%*" is the x coordinate of the upper left point of the image.

"*y%*" is the y coordinate of the upper left point of the image.

"*size_x%*" is the width of the image in pixels.

"*size_y%*" is the height of the image in pixels.

"*image$*" is a string containing the bitmap data of the image.

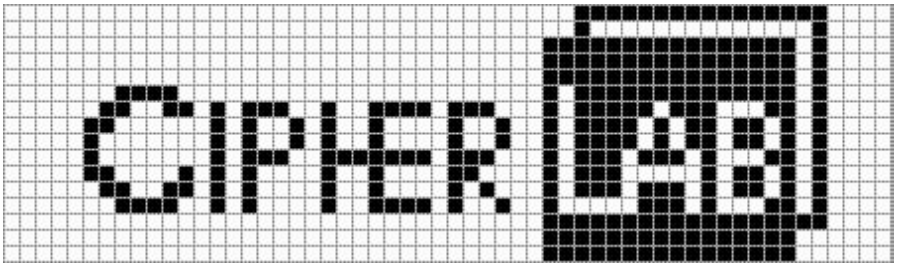
The user needs to allocate a string variable to store the bitmap data for the image. The string begins with the top row of pixels. Each row begins with the leftmost pixels. Each bit of the bitmap represents a single pixel

of the image. If the bit is set to 1, the pixel is marked, and if it is 0, the pixel is unmarked. The first pixel in each row is represented by the least significant bit of the first byte in each row. If the image is wider than 8 pixels, the ninth pixel in each row is represented by the least significant bit of the second byte in each row. Following is an example of the CipherLab company logo and the string variable--icon$, is used to store the bitmap data.

Usage

```
icon_1$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(248)+chr$(255)+chr$(7)
icon_2$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(8)+chr$(0)+chr$(4)
icon_3$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)
icon_4$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)
icon_5$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(5)
icon_6$ = chr$(192)+chr$(3)+chr$(0)+chr$(0)+chr$(250)+chr$(255)+chr$(5)
icon_7$ = chr$(96)+chr$(214)+chr$(201)+chr$(59)+chr$(250)+chr$(142)+chr$(5)
icon_8$ = chr$(48)+chr$(80)+chr$(74)+chr$(72)+chr$(122)+chr$(109)+chr$(5)
icon_9$ = chr$(16)+chr$(80)+chr$(74)+chr$(72)+chr$(122)+chr$(109)+chr$(5)
icon_10$ = chr$(16)+chr$(208)+chr$(249)+chr$(59)+chr$(186)+chr$(139)+chr$(5)
icon_11$ = chr$(48)+chr$(84)+chr$(72)+chr$(24)+chr$(58)+chr$(104)+chr$(5)
icon_12$ = chr$(96)+chr$(86)+chr$(72)+chr$(40)+chr$(186)+chr$(107)+chr$(5)
icon_13$ = chr$(192)+chr$(83)+chr$(200)+chr$(75)+chr$(130)+chr$(139)+chr$(5)
icon_14$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(7)
icon_15$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(1)
icon_16$ = chr$(0)+chr$(0)+chr$(0)+chr$(0)+chr$(254)+chr$(255)+chr$(1)

show_image(2, 0, 56, 1, icon_1$)
show_image(2, 1, 56, 1, icon_2$)
show_image(2, 2, 56, 1, icon_3$)
show_image(2, 3, 56, 1, icon_4$)
show_image(2, 4, 56, 1, icon_5$)
show_image(2, 5, 56, 1, icon_6$)
show_image(2, 6, 56, 1, icon_7$)
show_image(2, 7, 56, 1, icon_8$)
show_image(2, 8, 56, 1, icon_9$)
show_image(2, 9, 56, 1, icon_10$)
show_image(2, 10, 56, 1, icon_11$)
show_image(2, 11, 56, 1, icon_12$)
show_image(2, 12, 56, 1, icon_13$)
show_image(2, 13, 56, 1, icon_14$)
show_image(2, 14, 56, 1, icon_15$)
show_image(2, 15, 56, 1, icon_16$)
…
```



4.15 Battery Commands

This section describes the commands dealing with battery power.

BACKUP_BATTERY ~~201~~ ~~510~~ ~~520~~ **711** **720**

Purpose To get the voltage level of the backup battery.

Syntax *A*% = BACKUP_BATTERY

Remarks "*A%*" is an integer variable to be assigned with the voltage level of the backup battery.

The voltage level of the backup battery is returned in units of mili-volt (mV). The backup battery is used to backup the SRAM and keep the calendar chip running even when the system power is off.

The backup battery would be considered as "Battery Low" when the BACKUP_BATTERY is lower than 2900 (mV). That means the SRAM and calendar chip may lose their data at any time thereafter, if the battery is not recharged or replaced.

Usage

```
CheckBackupBattery:
    IF BACKUP_BATTERY < BATTERY_LOW% THEN
        BEEP(2000,30)
        CLS
        PRINT "Backup Battery needs to be replaced!"
        Loop:
            GOTO Loop
    END IF
```

MAIN_BATTERY ~~201~~ ~~510~~ ~~520~~ **711** **720**

Purpose To get the voltage level of the main battery.

Syntax *A*% = MAIN_BATTERY

Remarks "*A%*" is an integer variable to be assigned with the voltage level of the main battery.

The voltage level of main battery is returned in units of mili-volt (mV). The main battery is the power source for the system operation.

The main battery would be considered as "Battery Low" when the MAIN_BATTERY is lower than 2200(mV). That means the basic system operations may operate, yet some functions consuming high power may be disabled.

Usage

```
    BATTERY_LOW% = 2200
CheckMainBattery:
    IF MAIN_BATTERY < BATTERY_LOW% THEN
        BEEP(2000,30)
        CLS
        PRINT "Main Battery needs to be recharged!"
        Loop:
            GOTO Loop
    END IF
```

4.16 Communication Ports

There are three communication ports on 201, 510 & 520: COM1 and COM3 are for RS-232 communications, COM2 is for RS-485 communications.

There are two communication ports on 7xx terminals, namely COM1 and COM2. The user has to set up the communication type for the COM ports before they can be used. COM1 could be used for direct RS-232 or docking port. COM2 could be used for any of the following: Serial IR, IrDA, or RF port. The user needs to specify which type of interface is to be used, but can use the same commands to open, close, read, and write the data through to the different interfaces.

The communication related commands are divided into different types and are described separately as follows.

4.16.1 RS-232 Communications

(1) Baud Rate: One out of 8 baud rates can be selected.
520/7xx -- 115200, 76800, 57600, 38400, 19200, 9600, 4800, 2400
201/510 -- 38400, 19200, 9600, 4800, 2400, 1200, 600, 300

(2) Data Bits: 7 or 8

(3) Parity: Even, Odd or None

(4) Stop bit: 1

(5) Flow Control: RTS/CTS, XON/XOFF, or None

CLOSE_COM	**201**	**510**	**520**	**711**	**720**

Purpose To disable the specified communication port.

Syntax CLOSE_COM (*N%*)

Remarks "*N%*" is a numeric expression indicating which communication port is to be disabled.

Usage CLOSE_COM(3)

COM_DELIMITER	**201**	**510**	**520**	**711**	**720**

Purpose To change the delimiter of COM port sending & receiving string.

Syntax COM_DELIMITER (*N%, C%*)

Remarks "*N%*" is a numeric expression indicating which communication port is to be set. For 201/510, only COM3 is valid.

"C%", in the range of 0 to 255, stands for the ASCII code of the delimiter character. If it is negative, no delimiter will be applied.

Usage COM_DELIMITER(3, 10) REM use Line-feed as delimiter

GET_CTS	201	510	520	711	720

Purpose To get the CTS level from the specified communication port.

Syntax *A*% = GET_CTS (*N%*)

Remarks "*A*%" is an integer variable to be assigned with the result.

"*N%*" is a numeric expression indicating which communication port the user intends to get the CTS level for. The return value is 1 if the CTS is in "Mark" state; or 0, if the CTS is in "Space" state.

Usage A% = GET_CTS(1)

OPEN_COM	201	510	520	711	720

Purpose To enable the specified communication port.

Syntax OPEN_COM (*N%*)

Remarks "*N%*" is a numeric expression indicating which communication port is to be enabled.

Usage OPEN_COM(1)

READ_COM$	201	510	520	711	720

Purpose To read data from the specified communication port.

Syntax *A*$ = READ_COM$ (*N%*)

Remarks "*A*$" is a string variable to be assigned with the data.

"*N%*" is a numeric expression indicating from which communication port the data is to be read. If the buffer of the communication port is empty, an empty string will be returned.

Usage

```
ON COM(1) GOSUB HostCommand
…
HostCommand:
    Cmd$ = READ_COM$(1)
    CmdIdentifier$ = LEFT$(Cmd$, 1)
    DBFNum% = VAL(MID$(Cmd$,2,1))
    IDFNum% = VAL(MID$(Cmd$,3,1))
    CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
    IF CmdIdentifier$ = "-" THEN
        DEL_RECORD(DBFNum%, IDFNum%)
    ELSE
        …
```

SET_COM **201 510 520 711 720**

Purpose To set parameters for the specified communication port.

Syntax SET_COM$ (*N%, Baudrate%, Parity%, Data%, Handshake%*)

Remarks

Parameters	Values		Remarks
N%	1 - 3		Indicates which COM port is to be set.
Baudrate%	201/510	520/7xx	Specifies the baud rate of the COM port.
	1: 38400 bps 2: 19200 bps 3: 9600 bps 4: 4800 bps 5: 2400 bps 6: 1200 bps 7: 600 bps 8: 300 bps	1:115200 bps 2: 76800 bps 3: 57600 bps 4: 38400 bps 5: 19200 bps 6: 9600 bps 7: 4800 bps 8: 2400 bps	
Parity%	1: No Parity 2: Odd Parity 3: Even Parity		Specifies the parity of the COM port.
Data%	1: 7 Data Bits 2: 8 Data Bits		Specifies the data bits of the COM port.
Handshake%	1: No Handshake 2: CTS/RTS 3: XON/XOFF		Specifies the method of flow control for the COM port.

Usage SET_COM(1, 1, 1, 2, 1) ‘ COM1, 38400, N, 8, No handshake (for 201)

SET_COM_TYPE ~~**201 510 520**~~ **711 720**

Purpose To set the communication types for the COM ports.

Syntax SET_COM_TYPE (*Comport%, Type%*)

Remarks This function needs to be called before opening a COM port.

Comport%	Available Type
1	1: direct RS-232 2: docking (via communication cradle)
2	3: serial IR (via Serial IR Transceiver) 4: standard IrDA 5: RF communication

Usage SET_COM_TYPE(2, 3) ‘ set COM 2 to IR communication

SET_RTS	201	510	520	711	720

Purpose To set the CTS level for the specified communication port.

Syntax SET_RTS (*N%, State%*)

Remarks "*N%*" is a numeric expression indicating which communication port to set the CTS level for.

"*State%*" may be 1 or 0, indicates that the RTS state to be "Mark" or "Space" respectively.

Usage
```
SET_RTS(1, 1)                ' set COM1 RTS to "Mark"
```

WRITE_COM	201	510	520	711	720

Purpose To send data string to the host through the specified communication port.

Syntax WRITE_COM$ (*N%, Data$*)

Remarks "*N%*" is a numeric expression indicating which communication port the data is to be sent to.

"*Data$*" is the character string to be sent.

Usage
```
    ON READER(1) GOSUB BcrData_1
    …
BcrData_1:
    BEEP(2000, 5)
    Data$ = GET_READER_DATA$(1)
    WRITE_COM$(1, Data$)
    …
```

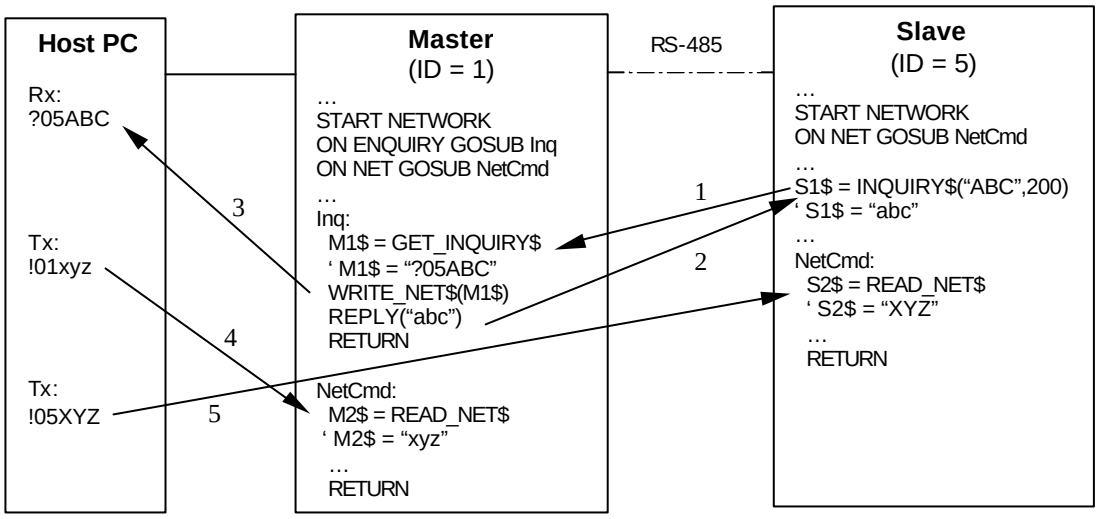
4.16.2 RS-485 Communications

This section describes the BASIC functions and statements related to the RS-485 network.

The RS-485 is a multi-drop communication standard which allows up to 32 stations (expandable by use of repeater) to be linked on the same network. Each station must be assigned with a unique station ID for proper communication. Only one of the stations on the network is assigned to be the bus arbitrator (master), while others are listeners (slaves). To avoid data collision, only the master can start communication to the slaves and only the slave whom the master calls can respond. (Slaves will not respond to broadcast messages.) With programming in BASIC, a valid station ID is a number in the range of 1 to 99, and the station with ID 1 is the master.

Commands READ_COM$ and WRITE_COM are used for the terminal to communicate with the host computer when the terminal is used in stand-alone mode. However, when the terminals are on the RS-485 network, it may be a little more involved. First, user has to call START NETWORK on each terminal to enable the RS-485 communications and then use different commands to exchange messages between the host computer and the terminals. Commands INQUIRY$, GET_INQUIRY$, and REPLY are used for the communications between the master and the slaves. Command WRITE_NET is used for the master to send messages to the host computer. Command READ_NET$ is used for the master and the slaves to receive the messages from the host computer. The message sent by the host computer must be in a format of "!ssx..x", where "!" is the command identifier, "ss" is a two-digit station ID of the destination terminal ("00" for broadcasting), and "x..x" is the message to be sent.

Let's take the system shown in the diagram as an example.



The BASIC command INQUIRY$("ABC", 200) on the slave will copy the inquiry identifier "?" and its station ID "05" and the data "ABC" to the RS-485 output buffer and wait for the response for 1 second. The data string in the RS-485 output buffer will be transmitted when the slave is polled by the master, and it will trigger the "INQUIRY Event" on the master.

On the master, the received data is assigned to M1$, and then sent to the host PC by the command WRITE_NET(M1$). So, the string "?05ABC" will be sent to the host PC. Then the master calls REPLY("abc") to reply the slave's inquiry. The string "abc" is returned for the command INQUIRY$("ABC") and assigned to S1$.

On the host PC, the user can send the string "!01xyz" through RS-232 and that will trigger the "NET Event" on the master. The command identifier "!" and the station ID in the received data will be stripped off by the background routines and then assigned to M2$ by the command M2$ = READ_NET$. The same, user can send the string "!05XYZ" through RS-232 and that will trigger the "NET Event" on the slave 05. S2$ is assigned with the string "XYZ" after the command S2$ = READ_NET$.

GET_INQUIRY$	201	510	520	~~711~~	~~720~~

Purpose To enable the master station to get the inquiry string from the slaves.

Syntax *A*$ = GET_INQUIRY$

Remarks "*A$*" is a string variable to be assigned with the inquiry from the slave which calls the command INQUIRY$.

Usage

```
    ON ENQUIRY GOSUB Verification
    …
Verification:
    LOCK
    InqString$ = GET_INQUIRY$
    InqData$ = RIGHT$(InqString$, LEN(InqString$) – 3)
    ‘ strip off the station ID and the inquiry identifier “?”
    FOR I = 1 TO 10
        WRITE_NET(InqData$)
        WAIT(100)    ‘ wait 0.5 sec
        Response$ = READ_NET$
        IF Response$ <> “” THEN GOTO ReplyInq
    NEXT I              ‘ retry
    Response$ = “Cannot find the Host!”
ReplyInq:
    REPLY(Response$)
    UNLOCK
    RETURN
    …
```

INQUIRY$	201	510	520	~~711~~	~~720~~

Purpose To send an inquiry string to the master and wait for response.

Syntax *A*$ = INQUIRY$ (*Inquiry$*, *Timeout*%)

Remarks "*A$*" is a string variable to be assigned with the response from the master.

"*Inquiry$*" is the inquiry string to be sent to the master.

"*Timeout*%" is an integer that indicates the time duration of the inquiry. It is specified in units of 5 ms. If no response is received after the time duration has expired, a NULL string will be returned.

Usage

```
    ON READER(1) GOSUB BcrData_1
    …
BcrData_1:
    BEEP(2000,5)
    BcrData$ = GET_READER_DATA$(1)
    Response$ = INQUIRY$(BcrData$)
    …
```

READ_NET$	201	510	520	~~711~~	~~720~~

Purpose To read messages from the host computer.

Syntax *A*$ = READ_NET$

Remarks "*A$*" is a string variable to be assigned with the message from the host computer.

The message sent by the host computer must be in a format of "!ssx..x", where "!" is the command identifier, "ss" is a two-digit station ID of the destination terminal ("00" for broadcasting), and "x..x" is the message to be sent.

Usage

```
    ON NET GOSUB NetCmd
    …
NetCmd:
    LOCK
    HostCmd$ = READ_NET$
    …
    UNLOCK
    RETURN
```

REPLY	201	510	520	~~711~~	~~720~~

Purpose To enable the master station to send a response string to the inquiry.

Syntax REPLY (*Reply$*)

Remarks "*Reply$*" is a character string to be sent back to the inquiry originator.

Usage

```
    ON ENQUIRY GOSUB Verfication
    …
Verfication:
    LOCK
    InqString$ = GET_INQUIRY$
    InqData$ = RIGHT$(InqString$, LEN(InqString$) – 3)
    ‘ strip off the station ID and the inquiry identifier “?”
    FOR I = 1 TO 10
        WRITE_NET(InqData$)
        WAIT(100)    ‘ wait 0.5 sec
        Response$ = READ_NET$
        IF Response$ <> “” THEN GOTO ReplyInq
    NEXT I              ‘ retry
    Response$ = “Cannot find the Host!”
ReplyInq:
    REPLY(Response$)
    UNLOCK
    RETURN
    …
```

START NETWORK	201	510	520	~~711~~	~~720~~

Purpose To enable the RS-485 communication.

Syntax START NETWORK

Remarks This command must be called before the other RS-485 related commands. After calling this command in a BASIC program, the user can issue a set of reserved RS-232 commands from the host computer to the master to access the data of the transaction file and also adjust the system time. Please refer to the section 4.20 for detailed information.

Usage START NETWORK

STOP NETWORK	201	510	520	~~711~~	~~720~~

Purpose To disable the RS-485 communication.

Syntax STOP NETWORK

Remarks This is the counter command of START NETWORK.

Usage STOP NETWORK

4.17 File Manipulation

This section describes the commands relating to file manipulation.

4.17.1 DAT Files

There are two different types of file structures supported in CipherLab BASIC. The first one is a sequential file structure, which is much like the ordinary sequential file but is modified to support FIFO structure. We call this type of file a DAT file. Because DAT files are usually used to store transaction data, they are also referred to as *Transaction* files. The length of the record in the transaction file is limited to 255 bytes.

For portable terminals 711 and 720, a BASIC program can have up to 6 transaction files. For fixed terminals 201, 510, and 520, only 1 transaction file can be used. When the fixed terminals are networked, the transaction data saved in the slaves will be automatically collected and saved in the master's transaction file and the original transaction data on the slaves will be erased.

DEL_TRANSACTION_DATA	**201**	**510**	**520**	**711**	**720**

Purpose To remove a block of transaction data from the default transaction file.

Syntax DEL_TRANSACTION_DATA (*N%*)

Remarks "*N%*" is a numeric expression determining how many transaction records to be deleted and how to delete.

If "N%" is a *positive* integer, the specified number of records will be deleted from the top of the transaction file 1, i.e., the oldest records will be deleted.

If "N%" is a *negative* integer, the specified number of records will be deleted from the bottom of the transaction file 1, i.e., the latest records will be deleted.

Usage

```
…
PRINT "Discard the latest transaction? (Y/N)"
…
Loop:
  KeyData$ = INKEY$
  IF KeyData$ = "" THEN
    GOTO Loop
  ELSE IF KeyData$ = "Y" THEN
    DEL_TRANSACTION_DATA(-1)
  END IF
  …
```

DEL_TRANSACTION_DATA_EX	~~201~~	~~510~~	~~520~~	**711**	**720**

Purpose To remove a block of transaction data from the specified transaction file.

Syntax DEL_TRANSACTION_DATA_EX (*file%, N%*)

Remarks "*file%*" is an integer in the range of 1 to 6 indicating which transaction file the command is to affect. The command DEL_TRANSACTION_DATA_EX(1, *N%*) works the same as the command DEL_TRANSACTION_DATA(*N%*).

"*N%*" is a numeric expression determining how many transaction records to be deleted and how to delete.

If “N%” is a positive integer, the specified number of records will be deleted from the top of the transaction file, i.e., the oldest records will be deleted.

If “N%” is a negative integer, the specified number of records will be deleted from the bottom of the transaction file, i.e., the latest records will be deleted.

Usage

```
…
PRINT “Discard the latest transaction? (Y/N)”
…
Loop:
  KeyData$ = INKEY$
  IF KeyData$ = “” THEN
    GOTO Loop
  ELSE IF KeyData$ = “Y” THEN
    DEL_TRANSACTION_DATA_EX(TransFile%,-1)
  END IF
  …
```

EMPTY_TRANSACTION **201 510 520 711 720**

Purpose To remove all the transaction data from the default transaction file.

Syntax EMPTY_TRANSACTION

Remarks This command will only have an effect on the first (default) transaction file.

Usage

```
…
PRINT “Remove all the transaction data? (Y/N)”
…
Loop:
  KeyData$ = INKEY$
  IF KeyData$ = “” THEN
    GOTO Loop
  ELSE IF KeyData$ = “Y” THEN
    EMPTY_TRANSACTION
  END IF
  …
```

EMPTY_TRANSACTION_EX ~~201~~ ~~510~~ ~~520~~ **711 720**

Purpose To remove all the transaction data from a specified transaction file.

Syntax EMPTY_TRANSACTION_EX (*file%*)

Remarks “*file%*” is an integer in the range of 1 to 6 indicating which transaction file the command is to affect. The command EMPTY_TRANSACTION_EX(1) works the same as the command EMPTY_TRANSACTION.

Usage EMPTY_TRANSACTION_EX (6)

GET_TRANSACTION_DATA$ **201 510 520 711 720**

Purpose To read a transaction record from the default transaction file.

Syntax *A*$ = GET_TRANSACTION_DATA$ (*N%*)

Remarks "*A$*" is a string variable to be assigned with the transaction data.

"*N%*" is a numeric expression indicating the ordinal number of the record to be read from the first transaction file.

Usage

```
…
WHILE TRANSACTION_COUNT > 0
    TransactionData$ = GET_TRANSACTION_DATA$
    WRITE_COM(1, TransactionData$)
    DEL_TRANSACTION_DATA(1)
    WEND
```

GET_TRANSACTION_DATA_EX$ ~~201~~ ~~510~~ ~~520~~ **711 720**

Purpose To read a transaction record from the specified transaction file.

Syntax *A*$ = GET_TRANSACTION_DATA_EX$ (*file%, N%*)

Remarks "*A$*" is a string variable to be assigned with the transaction data.

"*file%*" is an integer in the range of 1 to 6 indicating which transaction file to access. The command GET_TRANSACTION_DATA_EX$(1) works the same as the command GET_TRANSACTION_DATA$.

"*N%*" is a numeric expression indicating the ordinal number of the record to be read from the first transaction file.

Usage

```
…
WHILE (TRANSACTION_COUNT > 0)
    TransactionData$ = GET_TRANSACTION_DATA_EX$(TransFile%)
    WRITE_COM(1, TransactionData$)
    DEL_TRANSACTION_DATA_EX(TransFile%,1)
    WEND
```

SAVE_TRANSACTION **201 510 520 711 720**

Purpose To save (append) a transaction record to the default transaction file.

Syntax SAVE_TRANSACTION (*data$*)

Remarks "*data$*" is a string to be saved in the first (default) transaction file.

Usage

```
    ON READER(1) GOSUB BcrData_1
    …
BcrData_1:
    Data$ = GET_READER_DATA$(1)
    PRINT Data$
    SAVE_TRANSACTION(Data$)
    IF GET_FILE_ERROR <> 0 THEN PRINT "Transaction not saved."
    RETURN
```

SAVE_TRANSACTION_EX

~~201~~ ~~510~~ ~~520~~ **711** **720**

Purpose To save (append) a transaction record to the specified transaction file.

Syntax SAVE_TRANSACTION_EX(*file%*)

Remarks "*file%*" is an integer in the range of 1 to 6 indicating which transaction file to access. The command SAVE_TRANSACTION_EX(1) works the same as the command SAVE_TRANSACTION.

Usage

```
    ON READER(1) GOSUB BcrData_1
    …
BcrData_1:
    BEEP(2000,5)
    Data$ = GET_READER_DATA$(1)
    PRINT Data$
    SAVE_TRANSACTION_EX(TransFile%, Data$)
    IF GET_FILE_ERROR <> 0 THEN PRINT "Transaction not saved."
    RETURN
    …
```

TRANSACTION_COUNT

201 **510** **520** **711** **720**

Purpose To get the total number of transaction records saved in the default (first) transaction file.

Syntax *A*% = TRANSACTION_COUNT

Remarks "*A%*" is an integer variable to be assigned with the number of the transaction records.

Usage

```
    …
DataCount:
    DataCount% = TRANSACTION_COUNT
    CLS
    PRINT DataCount%, "transaction data is saved."
    RETURN
    …
```

TRANSACTION_COUNT_EX

~~201~~ ~~510~~ ~~520~~ **711** **720**

Purpose To get the total number of the transaction records saved in the specified transaction file.

Syntax *A*% = TRANSACTION_COUNT_EX (*file%*)

Remarks "*A%*" is an integer variable to be assigned with the number of the transaction records.

"*file%*" is an integer in the range of 1 to 6 indicating which transaction file to access. The command TRANSACTION_COUNT_EX(1) works the same as the command TRANSACTION_COUNT.

Usage

```
…
DataCount_1:
    DataCount% = TRANSACTION_COUNT_EX(1)
    CLS
    PRINT DataCount%, “data in transaction file 1.”
    RETURN
    …
```

UPDATE_TRANSACTION **201 510 520 711 720**

Purpose To update a transaction record in the default (first) transaction file.

Syntax UPDATE_TRANSACTION (*N%, data$*)

Remarks “*N%*“ is a numeric expression indicating the ordinal number of the transaction record to be updated.

“*data$*“ is the character string to replace the old data.

Usage

```
…
UpdateTransaction:
    UPDATE_TRANSACTION (Num%, NewData$)
    RETURN
    …
```

UPDATE_TRANSACTION_EX ~~201~~ ~~510~~ ~~520~~ **711 720**

Purpose To update a transaction record in the specified transaction file.

Syntax UPDATE_TRANSACTION_EX (*file%, N%, data$*)

Remarks “*file%*” is an integer in the range of 1 to 6 indicating which transaction file to access. The command UPDATE_TRANSACTION_EX(1, N%, Data$) works the same as the command UPDATE_TRANSACTION(N%, Data$).

“*N%*“ is a numeric expression indicating the ordinal number of the transaction record to be updated.

“*data$*“ is the character string to replace the old data.

Usage

```
…
UpdateTransaction_1:
    UPDATE_TRANSACTION_EX(1, Num%, NewData$)
    RETURN
    …
```

4.17.2 DBF Files and IDX Files

The other file structure supported is an index sequential file structure. Table look-up and report generation is easily supported by using index sequential file routines. There are actually two types of files associated with this file structure. These two types of files are called DBF files and IDX files.

A DBF file has a fixed record length structure. This is the file that stores the data records (members), whereas, the associated IDX files are the files that keep the information of the position of each record stored in the DBF file, but they are re-arranged (sorted) according to some specific key values. Besides the IDX files explicitly created by the user, the BASIC run-time also maintains a default IDX file which keeps the original data sequence.

A library would be a good example to illustrate how DBF and IDX files work. When you are trying to find a specific book in a library, you always start from the index. The book can be found by looking into the index of **book title**, **writer**, **publisher**, **ISBN number**, …etc. All these indexes are sorted in ascending order for easy lookup according to some specific information of books (book title, writer, publisher, ISBN number, …). When the book is found in the index, it will tell you where the book is actually stored.

As you can see, the books kept in the library are analogous to the data records stored in the DBF file, and the various indexes are just its associate IDX files. Some information in the data records (the book title, writer, publisher, and ISBN number) is used to create the IDX files.

A BASIC program can have up to 3 DBF files (5 DBF files for 7xx). Each DBF file can have up to 3 associated IDX files, and each of them is identified by its key (index) number. The valid key numbers are from 1 to 3. The length of the record in the DBF file is limited to 255 bytes.

Data records are not manipulated directly from the DBF file but rather through its associated IDX files. The value of file pointers of the IDX files (index pointers) does not represent the address of the data records stored in the DBF file. It indicates the sequence number of the specific data record in the IDX file.

ADD_RECORD	**201**	**510**	**520**	**711**	**720**

Purpose To add a record to the specified DBF file.

Syntax ADD_RECORD (*file%, data$*)

Remarks "*file%*" is an integer in the range of 1 to 3 (1 to 5 for 7xx) indicating which DBF file to be accessed.

"*data$*" is a character string which the user intends to add to the specified DBF file.

Usage

```
ON COM(1) GOSUB HostCommand
…
HostCommand:
    Cmd$ = READ_COM$(1)
    CmdIdentifier$ = LEFT$(Cmd$, 1)
    DBFNum% = VAL(MID$(Cmd$,2,1))
    CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-2)
    IF CmdIdentifier$ = "+" THEN
        ADD_RECORD(DBFNum%, CardID$)
    ELSE
        …
```

DEL_RECORD	201	510	520	711	720

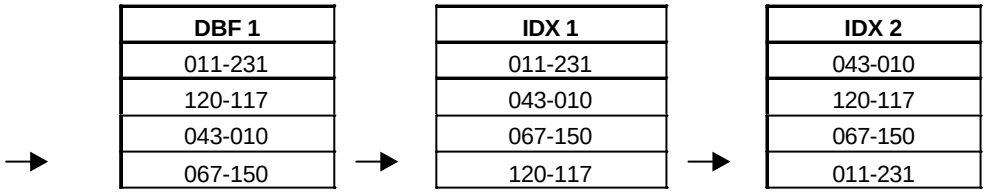
Purpose To delete the record pointed by the file pointer in the specified DBF file.

Syntax DEL_RECORD (*file%* [,*index%*])

Remarks "*file%*" is an integer in the range of 1 to 3 (1 to 5 for 7xx) indicating which DBF file to be accessed.

"*index%*" is an integer in the range of 1 to 3 indicating which IDX file to be accessed. If it is not specified, then the default IDX file which keeps the original data sequence will be used.

For example, if DBF 1 contains four records, 011-231,120-117, 043-010, 067-150. The key (index) of the first associate IDX is defined as starting at position 1 with length of 3, and the key (index) of the second associate IDX is defined as starting at position 5 with length of 3. All the file pointers of the DBF and IDX files are currently pointing to the last record.



DBF 1
011-231
120-117
043-010
067-150

IDX 1
011-231
043-010
067-150
120-117

IDX 2
043-010
120-117
067-150
011-231

Then, DEL_RECORD(1) will delete 067-150, DEL_RECORD(1,1) will delete 120-117, DEL_RECORD(1,2) will delete 011-231.

Usage

```
ON COM(1) GOSUB HostCommand
…
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$, 1)
  DBFNum% = VAL(MID$(Cmd$,2,1))
  IDFNum% = VAL(MID$(Cmd$,3,1))
  CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
  IF CmdIdentifier$ = "-" THEN
    DEL_RECORD(DBFNum%, IDFNum%)
  ELSE
    …
```

EMPTY_FILE	201	510	520	711	720

Purpose To remove all the records from the specified DBF file.

Syntax EMPTY_FILE (*file%*)

Remarks "*file%*" is an integer in the range of 1 to 3 (1 to 5 for 7xx) indicating which DBF file to be accessed.

Usage

```
ON COM(1) GOSUB HostCommand
…
HostCommand:
  Cmd$ = READ_COM$(1)
  CmdIdentifier$ = LEFT$(Cmd$, 1)
```

```
DBFNum% = VAL(MID$(Cmd$,2,1))
IDFNum% = VAL(MID$(Cmd$,3,1))
CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
IF CmdIdentifier$ = "!" THEN
   EMPTY_FILE(DBFNum%)
ELSE
…
```

FIND_RECORD	**201**	**510**	**520**	**711**	**720**

Purpose To search for a record in the specified DBF file, which matches the key string with respect to the specified IDX.

Syntax *A*% = FIND_RECORD (*file%, index%, key$*)

Remarks "*A%*" is an integer variable to be assigned with the result.

"*file%*" is an integer in the range of 1 to 3 (1 to 5 for 7xx) indicating which DBF file to be accessed.

"*index%*" is an integer in the range of 1 to 3 indicating which IDX file to be accessed.

"*key$*" is a character string which indicates the matching string to be found.

If any record member in the DBF file matches the key string with respect to the IDX, FIND_RECORD will return 1, and the file pointer of the IDX file will point to the first record with the matching string. If no match is found, the file pointer will point to the first record whose index value is greater than the value of "*key$*".

Usage

```
ON COM(1) GOSUB HostCommand
…
HostCommand:
   Cmd$ = READ_COM$(1)
   CmdIdentifier$ = LEFT$(Cmd$, 1)
   DBFNum% = VAL(MID$(Cmd$,2,1))
   IDXNum% = VAL(MID$(Cmd$,3,1))
   CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
   IF CmdIdentifier$ = "?" THEN
      IF FIND_RECORD(DBFNum%, IDXNum%, CardID$) = 1 THEN
            PRINT "Data is found in DBF", DBFNum%
      ELSE
            PRINT "Data is not found in DBF", DBFNum%
      END IF
   ELSE
      …
```

GET_RECORD$	**201**	**510**	**520**	**711**	**720**

Purpose To get a record in the specified DBF file, which is pointed to by the file pointer of the specified IDX file.

Syntax *A*$ = GET_RECORD$ (*file%* [*, index%*])

Remarks "*A$*" is a string variable to be assigned with the record.

"*file%*" is an integer in the range of 1 to 3 (1 to 5 for 7xx) indicating which DBF file to be accessed.

"*index%*" is an integer in the range of 1 to 3 indicating which IDX file to be accessed. If it is not specified, then the default IDX file which keeps the original data sequence will be used.

Usage

```
ON READER(1) GOSUB BcrData_1
…
BcrData_1:
    BEEP(2000,5)
    ID$ = GET_READER_DATA$(1)
    IF FIND_RECORD(DBFNum%, IDXNum%, ID$) = 1 THEN
        Data$ = GET_RECORD$(DBFNum%, IDXNum%)
        Item$ = MID$(Data$, IDLeng%+1, ItemLeng%)
        Note$ = RIGHT$(Data$, LEN(Data$)-IDLeng%-ItemLeng%)
        LOCATE 1,1
        PRINT "ID   :", Data$
        LOCATE 2,1
        PRINT "Item :", Item$
        LOCATE 3,1
        PRINT "Note :", Note$
    ELSE
        …
```

GET_RECORD_NUMBER

201 510 520 711 720

Purpose To get the ordinal number of the record pointed to by the file pointer of the specified DBF file and IDX file.

Syntax *A*% = GET_RECORD_NUMBER (*file%* [*, index%*])

Remarks "*A%*" is an integer variable to be assigned with the number.

"*file%*" is an integer in the range of 1 to 3 (1 to 5 for 7xx) indicating which DBF file to be accessed.

"*index%*" is an integer in the range of 1 to 3 indicating which IDX file to be accessed. If it is not specified, then the default IDX file which keeps the original data sequence will be used.

Usage A% = GET_RECORD_NUMBER (1, 1)

MOVE_TO

201 510 520 711 720

Purpose To move the file pointer, with respect to the specified DBF and IDX, to the specified position.

Syntax MOVE_TO (*file%,* [*index% ,*] *record_number%*)

Remarks "*file%*" is an integer in the range of 1 to 3 (1 to 5 for 7xx) indicating which DBF file to be accessed.

"*index%*" is an integer in the range of 1 to 3 indicating which IDX file to be accessed. If it is not specified, then the default IDX file which keeps the original data sequence will be used.

"*record_number%*" is a positive integer indicating the ordinal number of the record where the file pointer is moved to.

Usage MOVE_TO(1, 1, 20)

MOVE_TO_NEXT	**201**	**510**	**520**	**711**	**720**

Purpose To move the file pointer one record forward with respect to the specified DBF and IDX.

Syntax MOVE_TO_NEXT (*file%* [*, index%*])

Remarks "*file%*" is an integer in the range of 1 to 3 (1 to 5 for 7xx) indicating which DBF file to be accessed.

"*index%*" is an integer in the range of 1 to 3 indicating which IDX file to be accessed. If it is not specified, then the default IDX file which keeps the original data sequence will be used.

Usage MOVE_TO_NEXT(1,1)

MOVE_TO_PREVIOUS	**201**	**510**	**520**	**711**	**720**

Purpose To move the file pointer one record backward with respect to the specified DBF and IDX.

Syntax MOVE_TO_PREVIOUS (*file%* [*, index%*])

Remarks "*file%*" is an integer in the range of 1 to 3 (1 to 5 for 7xx) indicating which DBF file to be accessed.

"*index%*" is an integer in the range of 1 to 3 indicating which IDX file to be accessed. If it is not specified, then the default IDX file which keeps the original data sequence will be used.

Usage MOVE_TO_PREVIOUS(1,1)

RECORD_COUNT	**201**	**510**	**520**	**711**	**720**

Purpose To get the total number of the records in the specified DBF file.

Syntax *A*% = RECORD_COUNT (*file%*)

Remarks "*A%*" is an integer variable to be assigned with the number of the records in the DBF file.

"*file%*" is an integer in the range of 1 to 3 (1 to 5 for 7xx) indicating which DBF file to be accessed.

Usage TotalRecord_1% = RECORD_COUNT(1)

UPDATE_RECORD	**201**	**510**	**520**	**711**	**720**

Purpose To update the record pointed by the file pointer with respect to the specified DBF and IDX.

Syntax UPDATE_RECORD (*file%, index%, data$*)

Remarks "*file%*" is an integer in the range of 1 to 3 (1 to 5 for 7xx) indicating which DBF file to be accessed.

"*index%*" is an integer in the range of 1 to 3 indicating which IDX file to be accessed.

"*data$*" is the character string to replace the old data.

Usage

```
ON COM(1) GOSUB HostCommand
…
HostCommand:
    Cmd$ = READ_COM$(1)
    CmdIdentifier$ = LEFT$(Cmd$, 1)
    DBFNum% = VAL(MID$(Cmd$,2,1))
    IDXNum% = VAL(MID$(Cmd$,3,1))
    CardID$ = RIGHT$(Cmd$, LEN(Cmd$)-3)
    IF CmdIdentifier$ = “&” THEN
        UPDATE_RECORD(DBFNum%, IDXNum%, CardID$)
    ELSE
        …
```

4.17.3 Error Code

Command GET_FILE_ERROR returns the error code which is a number indicating the result of the last file manipulation. A value other than 0 indicates error.

GET_FILE_ERROR	201	510	520	711	720

Purpose To get the error code of the previous file manipulation command.

Syntax *A*% = GET_FILE_ERROR

Remarks "*A%*" is an integer variable to be assigned with the error code.

If there is no error, 0 will be returned. Possible error code and its interpretation is listed below.

Error Code	Interpretation
10	No free memory for file extension.

For other errors, such as invalid file ID, will cause a run-time error.

Usage

```
…
ADD_RECORD(1,Data$)
IF (GET_FILE_ERROR = 10) THEN
    ErrorMessage$ = "No free file space."
END IF
…
```

4.18 Memory

This section describes the commands concerning the flash memory, SRAM, and the Smart-Media Card (SMC).

FREE_MOMERY	**201**	**510**	**520**	**711**	**720**

Purpose To get the size of free data memory (SRAM) in bytes.

Syntax A& = FREE_MEMORY

Remarks "*A&*" is a long integer variable to be assigned with the result.

Usage PRINT "Free memory = ", FREE_MEMORY

RAM_SIZE	~~201~~	~~510~~	~~520~~	**711**	**720**

Purpose To get the size of data memory (SRAM) in Kilo-bytes.

Syntax A% = RAM_SIZE

Remarks "*A%*" is an integer variable to be assigned with the result.

Usage PRINT "SRAM size = ", RAM_SIZE

ROM_SIZE	~~201~~	~~510~~	~~520~~	**711**	**720**

Purpose To get the size of program memory (Flash) in Kilo-bytes.

Syntax A% = ROM_SIZE

Remarks "*A%*" is an integer variable to be assigned with the result.

Usage PRINT "Flash size = ", ROM_SIZE

SMC_FREE_MEMORY	~~201~~	~~510~~	~~520~~	~~711~~	**720**

Purpose To get free memory size of the Smart-Media Card in bytes.

Syntax A& = SMC_FREE_MEMORY

Remarks "*A&*" is a long integer variable to be assigned with the result.

Usage PRINT "SMC free memory = ", SMC_FREE_MEMORY

SMC_SIZE	~~201~~	~~510~~	~~520~~	~~711~~	**720**

Purpose To get the memory size of the Smart-Media Card in Kilo-bytes.

Syntax A% = SMC_SIZE

Remarks "*A%*" is an integer variable to be assigned with the result.

Usage PRINT "SMC memory size = ", SMC_SIZE

4.19 Debug Commands

Command START_DEBUG will write the activities happening on the system to the specified COM port. This command is very useful when the user wants to monitor the system or diagnose the problem.

When the command START_DEBUG is executed, the system will send a series of messages to the specified COM port until the command STOP_DEBUG is executed. The following are sample debug messages received when executing the sample program, 510AC_100.BAS. Please refer to section 5.4 for the source code of the program and to Appendix F for a description of the debug messages.

```
* L(7), T(0)
  ADD_RECORD(1,"10001Justin Jan
0830093011301300113015001800 2000")
* L(8), T(0)
* L(9), T(0)
  ASGN(2)
* L(10), T(0)
  ASGN(3)
* L(11), T(0)
  ASGN("CipherLab 510")
* L(12), T(0)
  ASGN("510AC_100.BAS")
* L(13), T(0)
…
* L(25), T(0)
  ARY(1)
  ASGN("OKGood Morning!       ")
…
* L(39), T(0)
  SET_COM(1,1,1,2,1)
* L(40), T(0)
  OPEN_COM(1)
…
* L(41), T(0)
  START_NETWORK
```

```
* L(42), T(0)
  ON_NET(316)
* L(43), T(0)
  ON_ENQUIRY(128)
…
  GOTO(68)
* L(68), T(0)
* L(69), T(0)
* L(70), T(0)
  GOTO(68)
…
* L(69), T(0)
  EVENT(16)
* L(79), T(1)
* L(80), T(1)
  OFF_READER(1)
* L(81), T(1)
  OFF_READER(2)
* L(82), T(1)
  CLS
* L(83), T(1)
  HIDE_CALENDAR
* L(84), T(1)
  BEEP(...)
```

START_DEBUG

201 510 520 711 720

Purpose To start the debug function.

Syntax START_DEBUG$ (*N%, Baudrate%, Parity%, Data%, Handshake%*)

Remarks

Parameters	Values		Remarks
N%	1 – 3		This parameter indicates which COM port to be used to send the debug messages.
Baudrate%	201/510	520/711/720	This parameter specifies the baud rate of the COM port.
	1: 38400 bps 2: 19200 bps 3: 9600 bps 4: 4800 bps 5: 2400 bps 6: 1200 bps 7: 600 bps 8: 300 bps	1:115200 bps 2: 76800 bps 3: 57600 bps 4: 38400 bps 5: 19200 bps 6: 9600 bps 7: 4800 bps 8: 2400 bps	
Parity%	1: No Parity 2: Odd Parity 3: Even Parity		This parameter specifies the parity of the COM port.
Data%	1: 7 Data Bits 2: 8 Data Bits		This parameter specifies the data bits of the COM port.
Handshake%	1: No Handshake 2: CTS/RTS 3: XON/XOFF		This parameter specifies the method of flow control of the COM port.

If a certain COM port has been used in the BASIC program, it is better to use another COM port for debugging to avoid conflicts.

Usage

```
START_DEBUG(1, 1, 1, 2, 1)
‘ use COM1 to send debug messages
‘ the COM port properties are 38400, N, 8, No handshake (for 201)
```

STOP_DEBUG

201 510 520 711 720

Purpose To terminate the debugging function.

Syntax STOP_DEBUG

Remarks This is the counter command of START_DEBUG.

Usage STOP_DEBUG

4.20 Reserved Host Commands

Usually, terminals on the RS-485 network use the command READ_NET$ to receive commands from the host computer, and they need to have functions in the BASIC program to process the received host commands. The host commands have a special format -- "!ssx..x", where "!" is the command identifier, "ss" is a two-digit station ID of the destination terminal ("00" for broadcasting), and "x..x" is the message to be sent.

After calling START NETWORK in BASIC program, user can issue a set of reserved RS-232 commands from the host computer to the master to inquire the on-line station IDs, to read/remove the data of the transaction file, or to adjust the system time. With the reserved RS-232 commands, the command identifier "!" and the station ID are not needed because these commands will be sent to the master and processed by the master only. The user's BASIC program does not need to do any processing because they will be processed by the background routines of the BASIC run-time.

Please note that each reserved RS-232 command is ended with a carriage return (can be changed by the COM_DELIMITER command). If any format error occurs, the master would echo "NAK".

CLEAR	**201**	**510**	**520**	**711**	**720**

Purpose To erase all the data of the transaction file.

Syntax CLEAR

CLEAR*N* -- for 7xx terminals only, where *N* is in the range of 1 to 6

Remarks The target terminal will delete all the data of the transaction file.

For 7xx terminals, there are up to 6 transaction files. User can specify which transaction file is to be erased.

Return OK, if the command is successfully processed.

NAK, if any format error occurs.

Usage CLEAR3 REM to delete data of the 3rd transaction file

READ	**201**	**510**	**520**	**711**	**720**

Purpose To read the top record of the transaction file.

Syntax READ

READ*N* -- for 7xx terminals only, where *N* is in the range of 1 to 6

Remarks The target terminal will return the top data of the transaction file back to the host computer.

For 7xx terminals, there are up to 6 transaction files. User can specify which transaction file is to be read.

Return OVER, if there is no data in the transaction file.

x..x, if the command is successfully processed, where "*x..x*" is the data string.

NAK, if any format error occurs.

Usage READ1 REM to read a record from the first transaction file

REMOVE	201	510	520	711	720

Purpose To remove one record from the top of the transaction file.

Syntax REMOVE

REMOVE*N* -- for 7xx terminals only, where *N* is in the range of 1 to 6

Remarks The target terminal will remove one record from the top of the transaction file.

For 7xx terminals, there are up to 6 transaction files. User can specify which transaction file is to be removed.

Return NEXT, if the command is processed successfully.

OVER, if no more data to be removed.

NAK, if any format error occurs.

Usage REMOVE2 REM to delete a record from the 2nd transaction file

TR	201	510	520	711	720

Purpose To get current system time.

Syntax TR

Remarks A string indicating the current system time will be returned to the host computer.

Return *yymmddhhnnss* / *yyyymmddhhnnss*, if the command is processed successfully, where

yy: last two digits of the year (201/510),

yyyy: year (520/711/720),

mm: month,

dd: day,

hh: hour (in 24-hour format)

nn: minute,

ss: second.

NAK, if any format error occurs.

Usage TR

TW	201	510	520	711	720

Purpose To set new system time.

Syntax TW*yymmddhhnnss* / TW*yyyymmddhhnnss*

where, yy: last two digits of the year (201/510)

yyyy: year (520/711/720),

mm: month,

dd: day,

hh: hour (in 24-hour format)

nn: minute,

ss: second

Remarks There is no station ID required. The master terminal will keep the system time synchronized among all terminals.

Return OK, if the command is successfully processed.
NAK, if any format error occurs.

Usage TW991201103000 ' for 201 / 510

TW19991201103000 ' for 520 / 711 / 720

X	**201**	**510**	**520**	~~711~~	~~720~~

Purpose To get the on-line station IDs.

Syntax X

Remarks The master will return all the station IDs on the RS-485 network to the host computer. This command is useful for system maintenance to see if the RS-485 communication is properly working or not.

Return *aa, bb, .., pp*, if the command is successfully processed, where "*aa, bb, pp*" are all two-digit station ID.
NAK, if any format error occurs.

Usage X

5 Sample Programs

This chapter provides six sample programs to illustrate the use of the BASIC commands.

5.1 201TEST_1.BAS

```
'*************************************************************************************************
' Program:          BASIC Sample Program -- 201TEST_1.BAS                                        *
' Target Machine:   201                                                                          *
' Description:      The terminal reads data from the reader ports and show the data and          *
'                   the code type on the LCD.                                                    *
'*************************************************************************************************
            'VERSION ("201TEST_1.BAS")
            'START_DEBUG(1,1,1,2,1)

'**********   Constants    **********
            BUZ_VOL% = 2
            BACK_LIT% = 3
            PROMPT1$ = "CipherLab 201"
            PROMPT2$ = "201TEST_1.BAS"

'**********  Initialization  **********
            ENABLE REA DER(1)
            ENABLE READER(2)
            ON READER(1) GOSUB BcrData_1
            ON READER(2) GOSUB BcrData_2
            Junk$ = GET_READER_DATA$(1)
            Junk$ = GET_READER_DATA$(2)

'**************************************
Main:
            VOL(BUZ_VOL%)
            BACKLIT(BACK_LIT%)
            BEEP(4750,5,0,5,4750,5,0,5,4750,5)
            CLS
            PRINT PROMPT1$
            LOCATE 2,1
            PRINT PROMPT2$
            LOCATE 2,14
            WAIT(400)
            CURSOR(0)
            LED(4,1,0)

Loop:

            GOTO Loop

'*************************************************************************************************
' Routine: BcrData_1                                                                             *
' Purpose:To get data from the reader port 1 and show it on the LCD.                             *
' Return:                                                                                        *
' Call:    CheckType                                                                             *
'*************************************************************************************************
BcrData_1:
            BEEP(4750,5)
            LED(1,1,2)
            BcrData$ = GET_READER_DATA$(1)
            GOSUB CheckType
            CLS
            PRINT BcrData$
            LOCATE 2,1
            PRINT BcrType$
            RETURN

'*************************************************************************************************
' Routine: BcrData_2                                                                             *
' Purpose:To get data from the reader port 2 and show it on the LCD.                             *
' Return:                                                                                        *
' Call:    CheckType                                                                             *
'*************************************************************************************************
BcrData_2:
            BEEP(4750,5)
            LED(1,1,2)
            BcrData$ = GET_READER_DATA$(2)
            GOSUB CheckType
```

```
        CLS
        PRINT BcrData$
        LOCATE 2,1
        PRINT BcrType$
        RETURN

'*************************************************************************************************
' Routine: CheckType                                                                             *
' Purpose:To check the type of the barcode.                                                      *
' Return:  BcrType$                                                                              *
' Call:                                                                                          *
'*************************************************************************************************
CheckType:
        IF (CODE_TYPE = "A") THEN
                BcrType$ = "Code 39"
        ELSE IF (CODE_TYPE = "B") THEN
                BcrType$ = "Italy Pharma-code"
        ELSE IF (CODE_TYPE = "C") THEN
                BcrType$ = "CIP 39"
        ELSE IF (CODE_TYPE = "D") THEN
                BcrType$ = "Industrial 25"
        ELSE IF (CODE_TYPE = "E") THEN
                BcrType$ = "Interleave 25"
        ELSE IF (CODE_TYPE = "F") THEN
                BcrType$ = "Matrix 25"
        ELSE IF (CODE_TYPE = "G") THEN
                BcrType$ = "Codabar (NW7)"
        ELSE IF (CODE_TYPE = "H") THEN
                BcrType$ = "Code 93"
        ELSE IF (CODE_TYPE = "I") THEN
                BcrType$ = "Code 128"
        ELSE IF (CODE_TYPE = "J") THEN
                BcrType$ = "UPCE no Addon"
        ELSE IF (CODE_TYPE = "K") THEN
                BcrType$ = "UPCE w/ Addon 2"
        ELSE IF (CODE_TYPE = "L") THEN
                BcrType$ = "UPCE w/ Addon 5"
        ELSE IF (CODE_TYPE = "M") THEN
                BcrType$ = "EAN8 no Addon"
        ELSE IF (CODE_TYPE = "N") THEN
                BcrType$ = "EAN8 w/ Addon 2"
        ELSE IF (CODE_TYPE = "O") THEN
                BcrType$ = "EAN8 w/ Addon 5"
        ELSE IF (CODE_TYPE = "P") THEN
                BcrType$ = "UPCA/EAN13 no Addon"
        ELSE IF (CODE_TYPE = "Q") THEN
                BcrType$ = "EAN13 w/ Addon 2"
        ELSE IF (CODE_TYPE = "R") THEN
                BcrType$ = "EAN13 w/ Addon 5"
        ELSE IF (CODE_TYPE = "S") THEN
                BcrType$ = "MSI"
        ELSE IF (CODE_TYPE = "T") THEN
                BcrType$ = "Plessey"
        ELSE IF (CODE_TYPE = "U") THEN
                BcrType$ = "Code ABC"
        ELSE IF (CODE_TYPE = "a") THEN
                BcrType$ = "ISO Track 1"
        ELSE IF (CODE_TYPE = "b") THEN
                BcrType$ = "ISO Track 2"
        ELSE IF (CODE_TYPE = "c") THEN
                BcrType$ = "ISO Track 1 and 2"
        ELSE IF (CODE_TYPE = "d") THEN
                BcrType$ = "ISO Track 2 and 3"
        ELSE
                BcrType$ = ""
        END IF
        RETURN

'##########  End of Program  ##########
```

5.2 201TEST_2.BAS

```
'*****************************************************************************************************
' Program:          BASIC Sample Program -- 201TEST_2.BAS                                            *
' Target Machine:   201                                                                              *
' Description:      The terminal reads data from the reader ports and shows it on the                *
'                   LCD for a period of time, then saves it to the transaction file. The             *
'                   saved data can be retrieved by the host PC. The system time is                   *
'                   shown when the terminal is not reading data and it is adjustable by              *
'                   an RS232 command from the host PC.                                               *
'*****************************************************************************************************
        'VERSION ("201TEST_2.BAS")
        'START_DEBUG(1,1,1,2,1)

'**********  Constants  **********
        BUZ_VOL% = 2
        BACK_LIT% = 3
        PROMPT1$ = "CipherLab 201"
        PROMPT2$ = "201TEST_2.BAS"
        INPUT_TIME_LENGTH% = 17
        NAK$ = "NAK"
        OVER$ = "OVER"
        _NEXT$ = "NEXT"             ' prefixed with an underline to distinguish from the reserved keyword NEXT
        DONE$ = "DONE"

'**********  Initialization  **********
        Set_Com(1,3,1,2,1) 'COM1, 9600, no parity, 8, no handshake
        OPEN_COM(1)
        ON COM(1) GOSUB ParseCmd
        ENABLE READER(1)
        ENABLE READER(2)
        ON READER(1) GOSUB BcrData_1
        ON READER(2) GOSUB BcrData_2
        Junk$ = GET_READER_DATA$(1)
        Junk$ = GET_READER_DATA$(2)

'*************************************
Main:
        VOL(BUZ_VOL%)
        BACKLIT(BACK_LIT%)
        BEEP(4750,5,0,5,4750,5,0,5,4750,5)
        CLS
        PRINT PROMPT1$
        LOCATE 2,1
        PRINT PROMPT2$
        LOCATE 2,14
        WAIT(400)
        CURSOR(0)
        LED(4,1,0)
        CLS
        SHOW_CALENDAR(1,1,1)
        BEEP(4750,5)

Loop:
        GOTO Loop

'*****************************************************************************************************
' Routine: BcrData_1                                                                                 *
' Purpose:To get data from the reader port 1 and show it on the LCD.                                 *
' Return:                                                                                            *
' Call:    SaveData                                                                                  *
'          Show Timer ON TIMER(1) -- 2 sec                                                           *
'*****************************************************************************************************
BcrData_1:
        OFF TIMER(1)
        BEEP(4750,5)
        LED(1,1,2)
        BcrData$ = GET_READER_DATA$(1)
        HIDE_CALENDAR
        CLS
        PRINT "Data:", BcrData$
        GOSUB SaveData
        ON TIMER(1,200) GOSUB Show Timer
        RETURN

'*****************************************************************************************************
' Routine: BcrData_2                                                                                 *
```

```
' Purpose:To get data from the reader port 2 and show it on the LCD.                    *
' Return:                                                                               *
' Call:       SaveData                                                                  *
'             Show Timer ON TIMER(1) -- 2 sec                                           *
'*****************************************************************************************
bcr2:
            OFF TIMER(1)
            BEEP(4750,5)
            LED(1,1,2)
            BcrData$ = GET_READER_DATA$(2)
            HIDE_CALENDAR
            CLS
            PRINT "Data:", BcrData$
            GOSUB SaveData
            ON TIMER(1,200) GOSUB Show Timer
            RETURN

'*****************************************************************************************
' Routine: SaveData                                                                      *
' Purpose:To save data to the transaction file.                                          *
' Return:                                                                               *
' Call:                                                                                 *
'*****************************************************************************************
SaveData:
            Data$ = Date$ + TIME$ + BcrData$
            SAVE_TRANSACTION(Data$)
            RETURN

'*****************************************************************************************
' Routine: Show Timer                                                                    *
' Purpose:To clean the screen and show timer.                                            *
' Return:                                                                               *
' Call:                                                                                 *
'*****************************************************************************************
Show Timer:
            OFF TIMER(1)
            SHOW_CALENDAR(1,1,1)
            RETURN

'*****************************************************************************************
' Routine: ParseCmd                                                                      *
' Purpose:To parse the host commands.                                                    *
' Return:                                                                               *
' Call:                                                                                 *
'*****************************************************************************************
ParseCmd:
            HostCmd$ = READ_COM$(1)
            IF (HostCmd$ = "READ") THEN
                        IF (TRANSACTION_COUNT > 0) THEN
                                    OutBuffer$ = GET_TRANSACTION_DATA$(1)
                        ELSE
                                    OutBuffer$ = OVER$
                        END IF
            ELSE IF (HostCmd$ = "REMOVE") THEN
                        DEL_TRANSACTION_DATA(1)
                        IF (TRANSACTION_COUNT > 0) THEN
                                    OutBuffer$ = _NEXT$
                        ELSE
                                    OutBuffer$ = OVER$
                        END IF
            ELSE IF (LEFT$(HostCmd$,4) = "TIME") THEN
                        IF (MID$(HostCmd$,5,1) = "=") AND (LEN(HostCmd$) = INPUT_TIME_LENGTH%) THEN
                                    DATE$ = MID$(HostCmd$,6,6)
                                    TIME$ = MID$(HostCmd$,12,6)
                                    OutBuffer$ = DONE$
                        ELSE
                                    OutBuffer$ = DATE$ + TIME$
                        END IF
            ELSE
                        OutBuffer$ = NAK$
            END IF
            WRITE_COM(1, OutBuffer$)
            RETURN

'##########  End of Program  ##########
```

5.3 201TEST_3.BAS

```
'*************************************************************************************************************
' Program:              BASIC Sample Program -- 201TEST_3.BAS                                                 *
' Target Machine:       201                                                                                   *
' Description:          The terminals read data from the reader ports and send it to the                      *
'                       master. The master collects data from slaves and send it to the PC.                   *
'                       The system time can be setup by the host PC.                                          *
'                       The data will be sent in a format of "ppYYYYMMDDhhmmssX..X",                          *
'                       where pp is the station ID where the data is read, X..X is the data.                  *
'                       This program can be used for the master and the slave.                                *
'*************************************************************************************************************
        'VERSION ("201TEST_3.BAS")
        'START_DEBUG(1,1,1,2,1)

'**********  Constants  **********
        BUZ_VOL% = 2
        BACK_LIT% = 3
        PROMPT1$ = "CipherLab 201"
        PROMPT2$ = "201TEST_3.BAS"
        INPUT_TIME_LENGTH% = 17
        NAK$ = "NAK"
        OVER$ = "OVER"
        _NEXT$ = "NEXT"             ' prefixed with an underline to distinguish from the reserved keyword NEXT
        DONE$ = "DONE"
        STATION_ID% = GET_STATION_ID

'**********  Initialization  **********
        SET_COM(1,3,1,2,1)          'COM1, 9600, no parity, 8, no handshake
        OPEN_COM(1)
        ON COM(1) GOSUB ParseCmd
        START NETWORK
        ENABLE READER(1)
        ENABLE READER(2)
        ON READER(1) GOSUB BcrData_1
        ON READER(2) GOSUB BcrData_2
        Junk$ = GET_READER_DATA$(1)
        Junk$ = GET_READER_DATA$(2)

'************************************
Main:
        VOL(BUZ_VOL%)
        BACKLIT(BACK_LIT%)
        BEEP(4750,5,0,5,4750,5,0,5,4750,5)
        CLS
        PRINT PROMPT1$
        LOCATE 2,1
        PRINT PROMPT2$
        LOCATE 2,14
        WAIT(400)
        CURSOR(0)
        LED(4,1,0)
        CLS
        SHOW_CALENDAR(1,1,1)
        BEEP(4750,5)

Loop:
        IF (STATION_ID% = 1) AND (TRANSACTION_COUNT > 0) THEN
                OutBuffer$ = GET_TRANSACTION_DATA$(1)
                DEL_TRANSACTION_DATA(1)
                WRITE_COM(1,OutBuffer$)
        END IF
        GOTO Loop

'*************************************************************************************************************
' Routine: BcrData_1                                                                                         *
' Purpose:To get data from the reader port 1 and show it on the LCD.                                         *
' Return:                                                                                                    *
' Call:         SaveData                                                                                     *
'               CleanScreen ON TIMER(1) -- 2 sec                                                             *
'*************************************************************************************************************
BcrData_1:
        OFF TIMER(1)
        BEEP(4750,5)
        LED(1,1,2)
        BcrData$ = GET_READER_DATA$(1)
        LOCATE 2,1
```

```
        PRINT "                 "
        LOCATE 2,1
        PRINT "Data:", BcrData$
        GOSUB SaveData
        ON TIMER(1,200) GOSUB CleanScreen
        RETURN

'*************************************************************************************************
' Routine: BcrData_2                                                                             *
' Purpose:To get data from the reader port 2 and show it on the LCD.                             *
' Return:                                                                                        *
' Call:    SaveData                                                                              *
'          CleanScreen ON TIMER(1) -- 2 sec                                                      *
'*************************************************************************************************
BcrData_2:
        OFF TIMER(1)
        BEEP(4750,5)
        LED(1,1,2)
        BcrData$ = GET_READER_DATA$(2)
        LOCATE 2,1
        PRINT "                 "
        LOCATE 2,1
        PRINT "Data:", BcrData$
        GOSUB SaveData
        ON TIMER(1,200) GOSUB CleanScreen
        RETURN

'*************************************************************************************************
' Routine: SaveData                                                                              *
' Purpose:To save data to the transaction file.                                                  *
' Return:                                                                                        *
' Call:                                                                                          *
'*************************************************************************************************
SaveData:
        IF STATION_ID% < 10 THEN
                Data$ = "0" + STR$(STATION_ID%) + DATE$ + TIME$ + BcrData$
        ELSE
                Data$ = STR$(STATION_ID%) + DATE$ + TIME$ + BcrData$
        END IF
        SAVE_TRANSACTION(Data$)
        RETURN

'*************************************************************************************************
' Routine: CleanScreen                                                                           *
' Purpose:To clean the screen.                                                                   *
' Return:                                                                                        *
' Call:                                                                                          *
'*************************************************************************************************
CleanScreen:
        OFF TIMER(1)
        LOCATE 2,1
        PRINT "                 "
        RETURN

'*************************************************************************************************
' Routine: ParseCmd                                                                              *
' Purpose:To parse the host commands.                                                            *
' Return:                                                                                        *
' Call:                                                                                          *
'*************************************************************************************************
ParseCmd:
        HostCmd$ = READ_COM$(1)
        IF (LEFT$(HostCmd$,4) = "TIME") THEN
                IF (MID$(HostCmd$,5,1) = "=") AND (LEN(HostCmd$) = INPUT_TIME_LENGTH%) THEN
                        DATE$ = MID$(HostCmd$,6,6)
                        TIME$ = MID$(HostCmd$,12,6)
                        OutBuffer$ = DONE$
                ELSE
                        OutBuffer$ = DATE$ + TIME$
                END IF
        ELSE
                OutBuffer$ = NA K$
        END IF
        WRITE_COM(1, OutBuffer$)
        RETURN

'##########  End of Program  ##########
```

5.4 510AC_100.BAS

```
'*************************************************************************************************************
'* Program:          BASIC Sample Program -- 510AC_100.BAS                                                   *
'* Target Machine:   510                                                                                     *
'* Description:      This is an access control program for 510. User can download data                       *
'*                   to DBF1 on the master by using the predefined RS-232 commands.                          *
'*                   The format of each record of the DBF1 is like                                           *
'*                       xxxxxyyyyyyyyyyyyyyyyaabbccddeeffgghhiijjkkllmmnnoopp, where*
'*                             - xxxxx: 5-digit employee ID                                                  *
'*                             - y..y: 15-digit employee name                                                *
'*                             - aabbccdd: the period of time for sign-in                                    *
'*                             - eeffgghh: the period of time for lunch break                                *
'*                             - iijjkkll:    the period of time for return from lunch break                 *
'*                             - mmnnoopp: the period of time for getting off                                *
'*                                                                                                           *
'*                   The commands for the host to manage the master, such as to adjust                       *
'*                   the timer or the modify DBF1 are listed as follows, assuming the                        *
'*                   station ID of the master is “01”.                                                       *
'*                             - “!01TIMER”: to inquire the current system time                              *
'*                             - “!01TIMER=yymmddhhnnss”: to set the current system time*
'*                             - “!01+xx..pp”: to add a record to the DBF1, where xx..pp is a*
'*                                   52-digits data for the employee, please refer to the*
'*                                   above description.                                                      *
'*                             - “!01-xx.pp”: to remove a record from the DBF1.                              *
'*                             -“!01?ALL”: to list all the records of the DBF1                               *
'*                             -“!01CIPHERLAB”: to tell the master that the connection                       *
'*                                   is still alive.                                                         *
'*                                                                                                           *
'*                   When an employee ID is read, the terminal will check if it is connected *
'*                   to the backend application by checking if the host PC keeps sending                     *
'*                   “!01CIPHERLAB\r” to the master consistently. If yes, it will let the back-*
'*                   end application to verify the ID; otherwise, the master will verify the ID *
'*                   according to the downloaded DBF1. The program will first verify the ID*
'*                   and then check the last event of the ID. If there is no last event                      *
'*                   recorded on the same day, the program will assume the employee is                       *
'*                   coming for the first event, i.e., to sign-in. If the ID is read within the              *
'*                   scheduled period of time, it will shows a greeting message on the LCD,*
'*                   otherwise it will show an error message on the LCD. All the data read*
'*                   will be saved in the transaction file with the station ID, date, time, and *
'*                   a log-flag. The log-flag ‘A’ indicates there was no connection between *
'*                   the master and the slave when data was read; ‘B’ indicates the data is*
'*                   an invalid data; ‘C’ indicates the data is a valid data; ‘D’ indicates any              *
'*                   communication error. User can use the reserved host commands to                         *
'*                   retrieve those data.                                                                    *
'*                                                                                                           *
'*                   DBF2 is used to save the last event for each ID read. The format of                     *
'*                   each record of the DBF2 is like                                                         *
'*                       xxxxxyymmddA, where                                                                 *
'*                             - xxxxx: 5-digit employee ID                                                  *
'*                             - yymmdd: date of the last event                                              *
'*                             - A: the last event                                                           *
'*                                                                                                           *
'*                   The DBF2 is on the master, and is only used by the master to verify                     *
'*                   the data. User does not access to DBF directly.                                         *
'*                                                                                                           *
'*************************************************************************************************************
        'VERSION("510AC_100.BAS")
        'START_DEBUG(1,1,1,2,1)
        'ADD_RECORD(1,"10001Justin Jan     0830093011301300113015001800 2000")          ' for test
'**********  Constants  **********
        BUZ_VOL% = 2
        BACK_LIT% = 3
        PROMPT1$ = "CipherLab 510"
        PROMPT2$ = "510AC_100.BAS"
        CONNECT1$ = "CIPHERLAB"
        CONNECT2$ = "SYNTECH"
        INPUT_TIME_LENGTH% = 17
        DATA_LENGTH% = 52 ' 5-digit ID + 15-digit Name + 32-digit Time (8X4 pairs)
        ID_LENGTH% = 5
        NAK$ = "NAK"
        OVER$ = "OVER"
        _NEXT$ = "NEXT"             ' prefixed with an underline to distinguish from the reserved keyword NEXT
        DONE$ = "DONE"
        NOMATCH$ = "NO MATCH"
        DIM MSG_OK$(4)
        DIM MSG_NG$(6)
```

```
            MSG_OK$(1) = "OKGood Morning!       "
            MSG_OK$(2) = "OKSee You Later!      "
            MSG_OK$(3) = "OKWelcome Back!       "
            MSG_OK$(4) = "OKHave A Nice Day!    "
            MSG_NG$(1) = "NGInvalid Entry Time! "
            MSG_NG$(2) = "NGInvalid Entry Time! "
            MSG_NG$(3) = "NGInvalid Entry Time! "
            MSG_NG$(4) = "NGInvalid Entry Time! "
            MSG_NG$(5) = "NGInvalid Entry Time! "
            MSG_NG$(6) = "NGInvalid Employee ID!"

            STATION_ID% = GET_STATION_ID

'********** Initialization  **********
            SET_COM(1,1,1,2,1)          'COM1, 38400, no parity, 8, no handshake
            OPEN_COM(1)
            START NETWORK
            ON NET GOSUB ParseCmd
            ON ENQUIRY GOSUB Verification
            ENABLE READER(1)
            ENABLE READER(2)
            Junk$ = GET_READER_DATA$(1)
            Junk$ = GET_READER_DATA$(2)
            ON READER(1) GOSUB BcrData_1
            ON READER(2) GOSUB BcrData_2
            ConnectFlag% = 0

'************************************
Main:
            VOL(BUZ_VOL%)
            BACKLIT(BACK_LIT%)
            BEEP(4750,5,0,5,4750,5,0,5,4750,5)
            CLS
            PRINT PROMPT1$
            LOCATE 2,1
            PRINT PROMPT2$
            CURSOR(0)
            WAIT(400)
            LED(4,1,0)
            CLS
            SHOW_CALENDAR(1,1,1)
            BEEP(4750,5)

Loop:

            GOTO Loop

'*****************************************************************************************************
' Routine: BcrData_1                                                                                 *
' Purpose:To get data from the reader port 1 and show it on the LCD.                                 *
' Return:                                                                                            *
' Call:    VerifyData                                                                                *
'          CleanScreen ON TIMER(1) -- 2 sec                                                          *
'*****************************************************************************************************
BcrData_1:
            OFF READER(1)
            OFF READER(2)
            CLS
            HIDE_CALENDAR
            BEEP(4750,5)
            LED(1,1,2)
            BcrData$ = GET_READER_DATA$(1)
            LOCATE 1,1
            PRINT "ID :", BcrData$
            GOSUB VerifyData
            ON TIMER(1,600) GOSUB ResetScreen
            Junk$ = GET_READER_DATA$(1)
            Junk$ = GET_READER_DATA$(2)
            ON READER(1) GOSUB BcrData_1
            ON READER(2) GOSUB BcrData_2
            RETURN

'*****************************************************************************************************
' Routine: BcrData_2                                                                                 *
' Purpose:To get data from the reader port 2 and show it on the LCD.                                 *
' Return:                                                                                            *
' Call:    SaveData                                                                                  *
'          CleanScreen ON TIMER(1) -- 2 sec                                                          *
```

```
'*************************************************************************************************
BcrData_2:
        OFF READER(1)
        OFF READER(2)
        CLS
        HIDE_CALENDAR
        BEEP(4750,5)
        LED(1,1,2)
        BcrData$ = GET_READER_DATA$(2)
        LOCATE 1,1
        PRINT "ID :", BcrData$
        GOSUB VerifyData
        ON TIMER(1,600) GOSUB ResetScreen
        Junk$ = GET_READER_DATA$(1)
        Junk$ = GET_READER_DATA$(2)
        ON READER(1) GOSUB BcrData_1
        ON READER(2) GOSUB BcrData_2
        RETURN

'*************************************************************************************************
' Routine: Verification                                                                          *
' Purpose:To verify data.                                                                        *
' Return:                                                                                        *
' Call:                                                                                          *
'*************************************************************************************************
Verification:
        LOCK
        InqString$ = GET_INQUIRY$
        InqData$ = RIGHT$(InqString$, LEN(InqString$) - 3)
                ' strip off the station ID & the inquiry identifier, "?".

        IF (ConnectFlag% = 1) THEN
                GOTO Verification_Host
        ELSE
                GOTO Verification_Master
        END IF

Verification_Host:
        FOR I = 1 To 10
                WRITE_NET (InqData$)
                WAIT(100)                                       ' wait 0.5 sec
                Response$ = READ_NET$
                IF Response$ <> "" THEN GOTO Host_Reply
        NEXT I

Host_Reply:
        REPLY(Response$)
        UNLOCK
        RETURN

Verification_Master:
        CurrentTime% = VAL(LEFT$(TIME$,4))  ' hhmm
        IF FIND_RECORD(1,1,InqData$) = 1 THEN
                OrgData$ = GET_RECORD$(1,1)
                Name$ = MID$(OrgData$,6,15)
                IF FIND_RECORD(2,1,InqData$) = 1 THEN
                        LastEvent$ = GET_RECORD$(2,1)
                        LastEventDate$ = MID$(LastEvent$,6,6)
                        IF LastEventDate$ = Date$ THEN
                                LastEventNo% = VAL(RIGHT$(LastEvent$,1))
                        ELSE
                                LastEventNo% = 0
                        END IF
                ELSE
                        LastEventNo% = 0
                END IF
                ON (LastEventNo%+1) GOSUB Event1, Event2, Event3, Event4, Event5

                IF ProcessFlag% = 1 THEN
                        EventData$ = InqData$ + Date$ + STR$(LastEventNo%)
                        'UPDATE_RECORD(2,1,InqData$)
                        FIND_RECORD(2,1,InqData$)
                        DEL_RECORD(2,1)
                        ADD_RECORD(2,EventData$)
                END IF
        ELSE
                Name$ = ""
                ReplyMsg$ = MSG_NG$(6)
```

```
        END IF

        REPLY(ReplyMsg$+Name$)
        UNLOCK
        RETURN


Event1:
        IF ( (CurrentTime% > VAL(MID$(OrgData$,21,4))) AND (CurrentTime% < VAL(MID$(OrgData$,25,4))) )
THEN
                ReplyMsg$ = MSG_OK$(1)
                ProcessFlag% = 1
        ELSE
                ReplyMsg$ = MSG_NG$(1)
                ProcessFlag% = 0
        END IF
        RETURN

Event2:
        IF (CurrentTime% > VAL(MID$(OrgData$,29,4)) AND (CurrentTime% < VAL(MID$(OrgData$,33,4))) THEN
                ReplyMsg$ = MSG_OK$(2)
                ProcessFlag% = 1
        ELSE
                ReplyMsg$ = MSG_NG$(2)
                ProcessFlag% = 0
        END IF
        RETURN

Event3:
        IF (CurrentTime% > VAL(MID$(OrgData$,37,4)) AND (CurrentTime% < VAL(MID$(OrgData$,41,4))) THEN
                ReplyMsg$ = MSG_OK$(3)
                ProcessFlag% = 1
        ELSE
                ReplyMsg$ = MSG_NG$(3)
                ProcessFlag% = 0
        END IF
        RETURN

Event4:
        IF (CurrentTime% > VAL(MID$(OrgData$,45,4)) AND (CurrentTime% < VAL(MID$(OrgData$,49,4))) THEN
                ReplyMsg$ = MSG_OK$(4)
                ProcessFlag% = 1
        ELSE
                ReplyMsg$ = MSG_NG$(4)
                ProcessFlag% = 0
        END IF
        RETURN

Event5:
        ReplyMsg$ = MSG_NG$(5)
        ProcessFlag% = 0
        RETURN


'*************************************************************************************************
' Routine: VerifyData                                                                            *
' Purpose:To verify data.                                                                        *
' Return:                                                                                        *
' Call:    SaveData                                                                              *
'*************************************************************************************************
VerifyData:
        Ans$ = INQUIRY$(BcrData$, 1000)                 ' Timeout = 5 sec
        IF Ans$ = "" THEN
                BEEP(4750,10,0,10,4750,10)
                MsgBuffer$ = "No Connection!"
                LogFlag$ = "A"
        ELSE IF LEFT$(Ans$, 2) = "NG" THEN
                BEEP(4750,10,0,10,4750,10)
                MsgBuffer$ = RIGHT$(Ans$, LEN(Ans$)-2)
                LogFlag$ = "B"
        ELSE IF LEFT$(Ans$, 2) = "OK" THEN
                BEEP(4750,5)
                MsgBuffer$ = RIGHT$(Ans$, LEN(Ans$)-2)
                LogFlag$ = "C"
        ELSE
                BEEP(4750,10,0,10,4750,10)
                MsgBuffer$ = "Communication error!"
                LogFlag$ = "D"
```

```
        END IF
        GOSUB ShowMsg
        GOSUB SaveData
        RETURN

'*************************************************************************************************
' Routine: ShowMsg                                                                               *
' Purpose:To show message.                                                                       *
' Return:                                                                                        *
' Call:                                                                                          *
'*************************************************************************************************
ShowMsg:
        IF LEN(MsgBuffer$) > 20 THEN
                LOCATE 1,1
                PRINT MsgBuffer$
                LOCATE 2,1
                PRINT MID$(MsgBuffer$,21,20)
        ELSE
                LOCATE 2,1
                PRINT MsgBuffer$
        END IF
        RETURN

'*************************************************************************************************
' Routine: SaveData                                                                              *
' Purpose:To save data to the transaction file.                                                  *
' Return:                                                                                        *
' Call:                                                                                          *
'*************************************************************************************************
SaveData:
        IF STATION_ID% < 10 THEN
                Data$ = "0" + STR$(STATION_ID%) + DATE$ + TIME$ + LogFlag$ + BcrData$
        ELSE
                Data$ = STR$(STATION_ID%) + DATE$ + TIME$ + LogFlag$ + BcrData$
        END IF
        SAVE_TRANSACTION(Data$)
        RETURN

'*************************************************************************************************
' Routine: ResetScreen                                                                           *
' Purpose:To clean the screen.                                                                   *
' Return:                                                                                        *
' Call:                                                                                          *
'*************************************************************************************************
ResetScreen:
        OFF TIMER(1)
        CLS
        SHOW_CALENDAR(1,1,1)
        RETURN

'*************************************************************************************************
' Routine: ParseCmd                                                                              *
' Purpose:To parse the host commands.                                                            *
' Return:                                                                                        *
' Call:                                                                                          *
'*************************************************************************************************
ParseCmd:
        LOCK
        HostCmd$ = READ_NET$
        IF LEFT$(HostCmd$,4) = "TIME" THEN
                IF MID$(HostCmd$,5,1) = "=" AND LEN(HostCmd$) = INPUT_TIME_LENGTH% THEN
                        DATE$ = MID$(HostCmd$,6,6)
                        TIME$ = MID$(HostCmd$,12,6)
                        OutBuffer$ = DONE$
                ELSE IF LEN(HostCmd$) = 4 THEN
                        OutBuffer$ = DATE$ + TIME$
                END IF
        ELSE
                CmdIdentifier$ = LEFT$(HostCmd$,1)
                EmpId$ = MID$(HostCmd$,2,5)
                IF CmdIdentifier$ = "+" AND (LEN(HostCmd$) = DATA_LENGTH% + 1) THEN
                        IF FIND_RECORD(1,1,EmpId$)=1 THEN
                                'UPDATE_RECORD(1,1,RIGHT$(HostCmd$,LEN(HostCmd$) - 1)
                                DEL_RECORD(1,1)
                                ADD_RECORD(1,RIGHT$(HostCmd$,LEN(HostCmd$) - 1))
                        ELSE
                                ADD_RECORD(1,RIGHT$(HostCmd$,LEN(HostCmd$) - 1))
                        END IF
```

```
                        OutBuffer$ = DONE$
                ELSE IF CmdIdentifier$ = "-" AND (LEN(HostCmd$) = ID_LENGTH% + 1) THEN
                        IF FIND_RECORD(1,1,EmpId$)=1 THEN
                                DEL_RECORD(1,1)
                                OutBuffer$ = DONE$
                        ELSE
                                OutBuffer$ = NOMATCH$
                        END IF
                ELSE IF HostCmd$ = "?ALL" THEN
                        MOVE_TO(1,1,1)
                        Temp$ = ""
Loop_Find:
                        OutBuffer$ = GET_RECORD$(1,1)
                        IF OutBuffer$ = Temp$ THEN
                                GOTO End_Find
                        ELSE
                                IF OutBuffer$ <> "" THEN WRITE_COM(1, OutBuffer$)
                                Temp$ = OutBuffer$
                                MOVE_TO_NEXT(1,1)
                        END IF
                        GOTO Loop_Find
End_Find:
                        OutBuffer$ = OVER$
                ELSE IF HostCmd$ = CONNECT1$ THEN
                        OutBuffer$ = CONNECT2$
                        GOSUB SetConnectFlag
                ELSE
                        OutBuffer$ = NAK$
                END IF
        END IF
        WRITE_NET(OutBuffer$)
        UNLOCK
        RETURN

'*************************************************************************************************
' Routine: SetConnectFlag                                                                        *
' Purpose:To set the connect flag.                                                               *
' Return:                                                                                        *
' Call:    ClearConnectFlag   - on TIMER2 (5 sec)                                                *
'*************************************************************************************************
SetConnectFlag:
        OFF TIMER (2)
        ConnectFlag% = 1
        ON TIMER (2,500) GOSUB ClearConnectFlag
        RETURN

'*************************************************************************************************
' Routine: ClearConnectFlag                                                                      *
' Purpose:To clear the connect flag.                                                             *
' Return:                                                                                        *
' Call:                                                                                          *
'*************************************************************************************************
ClearConnectFlag:
        OFF TIMER (2)
        ConnectFlag% = 0
        RETURN

'##########  End of Program  ##########
```

5.5 711AM_100.BAS

```
'*************************************************************************************************
' Program:          Archive Manager                                                              *
' Target Machine:   711                                                                          *
' Description:      This is a sample Basic program illustrates the use of the Basic              *
'                   statements for 711 terminals.                                                *
'                                                                                                *
'                   After power on, user needs to press F1 to login his ID before reading        *
'                   data. The user IDs in DBF1 can be inserted/deleted by RS-232                 *
'                   commands from the host computer. The user's IDs and passwords                *
'                   has to be matched with the records in DBF1 for a successful login.           *
'                   However, the password is not required for the first time login.              *
'                   User can press F3 to define/change his password.                             *
'                   User can press F2 and then key '9' to logout.                                *
'                   User will be forced to logout IF no action is taken in 10 min.               *
'                   Only labels terminated with 'A' or 'B' can be accepted, others will          *
'                   result in the "Invalid Data." error message. Once label 'A' is scanned,      *
'                   label 'B' with the same data is required for a successful match reading.     *
'                   Scanning the same label twice will result in the "Wrong label." error        *
'                   message. All results will be recorded to DBF2. The supervisor can            *
'                   read all records or only records related to a certain user by RS-232         *
'                   commands '?'. Please refer to routine HostCmd for details.                   *
'                                                                                                *
'           COM Port Settings: 38400 baud, 8 data bits, no parity, no handshake, 1 stop bit      *
'                                                                                                *
'           Function Keys:      F1 (FN + 1) : login                                          *
'                               F2 (FN + 2) : logout                                         *
'                               F3 (FN + 3) : change user's password                         *
'                                                                                                *
'           Database Files:     DBF1 : xxxxxpp..pp                                           *
'                               DBF2 : yyyymmddhhnnssxxxxxr[zz..zz]                          *
'                               xxxxx     -- 5 digit user ID                                 *
'                               pp..pp    -- length-undefined user password                 *
'                               yyyy      -- year                                            *
'                               mmdd      -- date                                            *
'                               hhnnss    -- time                                            *
'                               r         -- Error Index defined in routines ReadData,       *
'                                            Login, Logout and ForceToLogout,                  *
'                                                0 -- successful match                       *
'                                                1 -- wrong label                            *
'                                                2 -- invalid data                           *
'                                                A -- successful login                       *
'                                                B -- fail to login (invalid password)       *
'                                                C -- fail to login (invalid ID)             *
'                                                D -- logout                                 *
'                                                E -- forced to logout                       *
'                                                                                            *
'                               zz..zz    -- scanned data, if available                      *
'                                                                                            *
'*************************************************************************************************
        'VERSION ("711AM_100.BAS")
        'START_DEBUG(1,1,1,2,1)

'**********  Constants  **********
        PROMPT1$ = "CipherLab 711"
        PROMPT2$ = " Archive Manager "

'**********  Initialization  **********
        SET_COM(1,1,1,2,1)          'COM1: 38400, no parity, 8, no handshake
        OPEN_COM(1)
        ON COM(1) GOSUB HostCmd
        ENABLE READER(1)
        Junk$ = GET_READER_DATA$(1)

'************************************
Main:

        BACKLIT(BACK_LIT%)
        BEEP(2000,20,0,10,2000,10,0,10,2000,10)
        LOCATE 1, 6
        PRINT PROMPT1$
        LOCATE 2, 4
        PRINT PROMPT2$
        WAIT(400)
        CURSOR(0)
```

```
Loop:
        IF LoginFlag% = 1 THEN
                OFF KEY(1)
                ON KEY(2) GOSUB Logout
                ON KEY(3) GOSUB ChgPasswd
                ON READER(1) GOSUB ReadData
        ELSE
                OFF READER(1)
                OFF KEY(2)
                OFF KEY(3)
                ON KEY(1) GOSUB Login
        END IF

        Junk$ = INKEY$
        Junk$ = GET_READER_DATA$(1)
        GOTO Loop

'*************************************************************************************************
' Routine: Login                                                                                 *
' Purpose:To log in a new user.                                                                  *
' Return:  LoginFlag% :     1 -- login succeeds                                                  *
'                           0 -- login fails                                                     *
'          ErrorIndex$ :    A -- login succeeds                                                  *
'                           B -- login fails, invalid password                                   *
'                           C -- login fails, invalid ID                                         *
' Call:    ClearLCD on timer(1)                                                                 *
'          ForceToLogout on timer(2)                                                            *
'          WriteToFile                                                                          *
'*************************************************************************************************
Login:
        BEEP(3000,10,0,10,3000,10)
        OFF TIMER(1)
        CLS
        LOCATE 1,1
        PRINT "User ID : ";
        INPUT_MODE(1)
        INPUT UserID$
        IF FIND_RECORD(1,1,UserID$) = 1 THEN
                LOCK
                User$ = GET_RECORD$(1,1)
                UNLOCK
                UserPasswd$ = MID$(User$,6,6)
                IF UserPasswd$ = "" THEN GOTO Login_Succeed
                BEEP(3000,10,0,10,3000,10)
                LOCATE 2,1
                PRINT "Password : "

Loop_Login:
                INPUT_MODE(2)
                INPUT Password$
                IF Password$ = "" THEN GOTO Loop_Login

                IF Password$ = UserPasswd$ THEN

Login_Succeed:
                        LoginFlag% = 1
                        BEEP(3000,30)
                        LOCATE 3,1
                        PRINT "Login succeeded!"
                        ON TIMER(1,200) GOSUB ClearLCD          ' TIMER(1): 2 sec
                        ON TIMER(2,60000) GOSUB ForceToLogout   ' TIMER(2): 10 min
                        ErrorIndex$ = "A"
                        GOTO Exit_Login
                ELSE
                        BEEP(2000,10,3000,10,2000,10,3000,10)
                        LOCATE 3,1
                        PRINT "Invalid Password!"
                        ErrorIndex$ = "B"
                END IF

        ELSE
                BEEP(2000,10,3000,10,2000,10,3000,10)
                LOCATE 2,1
                PRINT "Invalid ID!"
                ErrorIndex$ = "C"
        END IF

        LoginFlag% = 0
```

```
Exit_Login:
        Data$ = ""
        GOSUB WriteToFile
        ON TIMER(1,200) GOSUB ClearLCD      ' TIMER(1): 2 sec
        RETURN

'*************************************************************************************************
' Routine: Logout                                                                                *
' Purpose:To log out the current user.                                                           *
' Return:  LoginFlag% :        0 -- logout succeeds                                              *
'          ErrorIndex$ :       D -- logout succeeds                                              *
' Call:    ClearLCD on timer(1)                                                                  *
'          ForceToLogout on timer(2)                                                             *
'*************************************************************************************************
Logout:
        IF LoginFlag% = 0 THEN RETURN
        BEEP(3000,10,0,10,3000,10)
        OFF TIMER(1)
        OFF TIMER(2)
        CLS
        LOCATE 1,1
        PRINT "User ID : "; UserID$
        LOCATE 2,1
        PRINT "Press '9' to Logout. "

Loop_Logout:
        B$ = INKEY$
        IF B$ = "" THEN
                GOTO Loop_Logout

        ELSE IF B$ = "9" THEN
                Data$ = ""
                ErrorIndex$ = "D"
                GOSUB WriteToFile
                LoginFlag% = 0
                ErrorIndex$ = "0"
                ScanFlag% = 0
                UserID$ = ""
                Password$ = ""
                NewPassword$ = ""
                TempData$ = ""
                PreviousRecord$ = ""
                TempRecord$ = ""
                CLS
                LOCATE 2,10
                PRINT "Bye. "
                BEEP(3000,30)
                ON TIMER(1,200) GOSUB ClearLCD              ' TIMER(1): 2 sec
        ELSE
                LOCATE 3,1
                PRINT "Go back to work. "
                BEEP(2000,20)
                ON TIMER(1,200) GOSUB ClearLCD              ' TIMER(1): 2 sec
                ON TIMER(2,60000) GOSUB ForceToLogout       ' TIMER(2): 10 min
        END IF
        RETURN

'*************************************************************************************************
' Routine: ChgPasswd                                                                             *
' Purpose:To change the user's password.                                                         *
' Return:  LoginFlag% :        0 -- system error                                                 *
' Call:    ClearLCD on timer(1)                                                                  *
'          ForceToLogout on timer(2)                                                             *
'*************************************************************************************************
ChgPasswd:
        IF LoginFlag% = 0 THEN RETURN
        BEEP(3000,10,0,10,3000,10)
        OFF TIMER(1)
        OFF TIMER(2)
        CLS
        LOCATE 1,1
        PRINT "User ID : "; UserID$
        LOCATE 2,1
        PRINT "Enter new password : "
        LOCATE 3,1

Loop_ChgPasswd1:
```

```
        INPUT_MODE(2)
        INPUT NewPassword1$
        IF NewPassword1$ = "" THEN GOTO Loop_ChgPasswd1

        CLS
        LOCATE 1,1
        PRINT "User ID : "; UserID$
        LOCATE 2,1
        PRINT "Please enter again."
        LOCATE 3,1

Loop_ChgPasswd2:
        INPUT_MODE(2)
        INPUT NewPassword2$
        IF NewPassword2$ = "" THEN GOTO Loop_ChgPasswd2

        IF NewPassword1$ = NewPassword2$ THEN
                LOCK
                IF FIND_RECORD(1,1,UserID$) = 1 THEN
                        A$ = UserID$ + NewPassword1$
                        DEL_RECORD(1,1)
                        ADD_RECORD(1,A$)
                        BEEP(3000,30)
                        LOCATE 4,1
                        PRINT "New password saved. "
                ELSE
                        BEEP(2000,10,3000,10,2000,10,3000,10)
                        LOCATE 4,1
                        PRINT "System Error."
                        LoginFlag% = 0
                END IF
                UNLOCK

        ELSE
                BEEP(2000,10,3000,10,2000,10,3000,10)
                CLS
                LOCATE 1,1
                PRINT "User ID : "; UserID$
                LOCATE 2,1
                PRINT "Not matched."
        END IF

        ON TIMER(1,200) GOSUB ClearLCD                  ' TIMER(1): 2 sec
        ON TIMER(2,60000) GOSUB ForceToLogout           ' TIMER(2): 10 min
        RETURN

'*************************************************************************************************
' Routine: ForceToLogout                                                                         *
' Purpose:To force the current user to logout, IF no action in 1 min.                            *
' Return:  LoginFlag% :        0 -- logout succeeds                                              *
'          ErrorIndex$ :       E -- forced to logout                                             *
' Call:    WriteToFile                                                                           *
'*************************************************************************************************
ForceToLogout:
        BEEP(3000,30)
        OFF TIMER(1)
        OFF TIMER(2)
        ErrorIndex$ = "E"
        Data$ = ""
        GOSUB WriteToFile
        CLS
        LoginFlag% = 0
        RETURN


'*************************************************************************************************
' Routine: HostCmd                                                                               *
' Purpose:To process the commands from the host computer,                                        *
'                 "+"xxxxx --> insert new user ID or rewrite old ID;                             *
'                 "-"xxxxx  --> delete old user ID;                                              *
'                 "List"          --> list all user IDs;                                         *
'                 "?"xxxxx --> inquire a certain user's record;                                  *
'                 "?All"          --> list all records;                                          *
'                 "<"             --> move the DBF2 pointer to previous;                         *
'                 ">"             --> move the DBF2 pointer to next;                             *
'                 "Read"          --> read the record pointed by the pointer;                    *
'                 "Remove"--> delete the record pointed by the pointer;                          *
'                 where xxxxx means 5-digit user ID.                                             *
```

```
' Return:                                                                                                    *
' Call:                                                                                                      *
'*************************************************************************************************************
HostCmd:
        ComCommand$ = READ_COM$(1)

        IF ComCommand$ = "" THEN
                WRITE_COM(1, "NAK")
                RETURN
        END IF

        CmdLength%  = LEN(ComCommand$)
        Command$ = LEFT$(ComCommand$,1)
        CmdData$ = RIGHT$(ComCommand$, LEN(ComCommand$) - 1)

        '////////////////////////      +  (Add user ID)      ////////////////////////////
        IF Command$ = "+" THEN
                IF CmdLength% <> 6 THEN
                        WRITE_COM(1, "NAK")
                ELSE IF FIND_RECORD(1,1,MID$(CmdData$,1,5)) = 0 THEN
                        ADD_RECORD(1,CmdData$)
                        WRITE_COM(1, "DONE")
                ELSE
                        DEL_RECORD(1,1)
                        ADD_RECORD(1,CmdData$)
                        WRITE_COM(1, "DONE")
                END IF

        '////////////////////////         -  (Delete user ID) ////////////////////////////
        ELSE IF Command$ = "-" THEN
                IF CmdLength% <> 6 THEN
                        WRITE_COM(1, "NAK")
                ELSE IF FIND_RECORD(1, 1, MID$(CmdData$,1,5)) = 1 THEN
                        DEL_RECORD(1,1)
                        WRITE_COM(1, "DONE")
                ELSE
                        WRITE_COM(1, "NO MATCH")
                END IF

        '////////////////////////     LIST  (List all user IDs)             ////////////////////////////
        ELSE IF ComCommand$ = "List" THEN
                MOVE_TO(1,1,1)
Loop_LIST:
                Record$ = GET_RECORD$(1,1)

                ' only user's ID to be listed, exclude the password
                Record$ = LEFT$(Record$,5)

                IF Record$ = "" THEN
                        WRITE_COM(1, "VOID")
                ELSE
                        WRITE_COM(1, Record$)
                END IF

                MOVE_TO_NEXT(1,1)
                TempRecord$ = GET_RECORD$(1,1)

                ' only user's ID to be listed, exclude the password
                TempRecord$ = LEFT$(TempRecord$,5)

                IF TempRecord$ <> Record$ THEN GOTO Loop_LIST

        '////////////////////////     ?All  (List all records)////////////////////////////
        ELSE IF ComCommand$ = "?All" THEN
                MOVE_TO(2,2,1)
Loop_ALL:
                Record$ = GET_RECORD$(2,2)
                IF Record$ = "" THEN
                        WRITE_COM(1, "VOID")
                ELSE
                        WRITE_COM(1, Record$)
                END IF
                MOVE_TO_NEXT(2,2)
                TempRecord$ = GET_RECORD$(2,2)
                IF TempRecord$ <> Record$ THEN GOTO Loop_ALL

        '////////////////////////     ?  (Find user's record)             ////////////////////////////
        ELSE IF Command$ = "?" THEN
```

```
                IF CmdLength% <> 6 THEN
                        WRITE_COM(1, "NAK")
                        RETURN
                ELSE
                        MOVE_TO(2,2,1)
Loop_Find:
                        Record$ = GET_RECORD$(2,2)
                        IF MID$(Record$,15,5) = CmdData$ THEN WRITE_COM(1, Record$)
                        IF TempRecord$ = Record$ THEN GOTO Next_Find
                        TempRecord$ = Record$
                        MOVE_TO_NEXT(2,2)
                        GOTO Loop_Find
                END IF
Next_Find:
                WRITE_COM(1, "OVER")

        '////////////////////////    <  (Move to previous]//////////////////////////
        ELSE IF ComCommand$ = "<" THEN
                MOVE_TO_PREVIOUS(2,1)
                WRITE_COM(1, "DONE")

        '////////////////////////         >  (Move to next]  //////////////////////////
        ELSE IF ComCommand$ = ">" THEN
                MOVE_TO_NEXT(2,1)
                WRITE_COM(1, "DONE")

        '////////////////////////       Read  (read record]//////////////////////////
        ELSE IF ComCommand$ = "Read" THEN
                Record$ = GET_RECORD$(2,1)
                IF Record$ = "" THEN
                        WRITE_COM(1, "VOID")
                ELSE
                        WRITE_COM(1, Record$)
                END IF

        '////////////////////////    Remove (delete record]             //////////////////////////
        ELSE IF ComCommand$ = "Remove" THEN
                Record$ = GET_RECORD$(2,1)
                IF Record$ = "" THEN
                        WRITE_COM(1, "VOID")
                ELSE
                        DEL_RECORD(2,1)
                        WRITE_COM(1, "DONE")
                END IF

        '////////////////////////      other commands     //////////////////////////
        ELSE
                WRITE_COM(1, "NAK")
        END IF

        RETURN

'*************************************************************************************************
' Routine: ReadData                                                                              *
' Purpose:To get data from the scanner port and show it on the LCD.                              *
' Return:  ErrorIndex$ :        0 -- successful match                                            *
'                               1 -- wrong label                                                 *
'                               2 -- invalid data                                                *
' Call:    WriteToFile                                                                           *
'          ClearLCD on timer(1)                                                                  *
'          ForceToLogout on timer(2)                                                             *
'*************************************************************************************************
ReadData:
        Data$ = GET_READER_DATA$(1)
        OFF TIMER(1)
        OFF TIMER(2)
        IF RIGHT$(Data$,1) = "A" THEN
                IF ScanFlag% = 0 THEN
                        ScanFlag% = 1
                        BEEP(3000,10)
                        CLS
                        PRINT Data$
                        TempData$ = LEFT$(Data$,LEN(Data$)-1)
                        PRINT "Please scan 'B'."
                        GOTO Exit_ReadData

                ELSE IF (ScanFlag% = 2) THEN
                        Data$ = LEFT$(Data$,LEN(Data$)-1)
```

```
                        IF Data$ = TempData$ THEN
                                BEEP(3000,30)
                                CLS
                                PRINT "OK!"
                                ErrorIndex$ = "0"               ' Successful match.
                        END IF
                ELSE
                        BEEP(2000,10,3000,10,2000,10,3000,10)
                        CLS
                        PRINT "Wrong label."
                        ErrorIndex$ = "1"                       ' The labels does not match.
                END IF

        ELSE IF RIGHT$(Data$,1) = "B" THEN
                IF ScanFlag% = 0 THEN
                        ScanFlag% = 2
                        BEEP(3000,10)
                        CLS
                        PRINT Data$
                        TempData$ = LEFT$(Data$,LEN(Data$)-1)
                        PRINT "Please scan 'A'."
                        GOTO Exit_ReadData

                ELSE IF (ScanFlag% = 1) THEN
                        Data$ = LEFT$(Data$,LEN(Data$)-1)
                        IF Data$ = TempData$ THEN
                                BEEP(3000,30)
                                CLS
                                PRINT "OK!"
                                ErrorIndex$ = "0"               ' Successful match.
                        END IF
                ELSE
                        BEEP(2000,10,3000,10,2000,10,3000,10)
                        CLS
                        PRINT "Wrong label."
                        ErrorIndex$ = "1"                       ' The labels does not match.
                END IF

        ELSE
                BEEP(2000,10,3000,10,2000,10,3000,10)
                CLS
                PRINT "Invalid Data."
                ErrorIndex$ = "2"                               ' Invalid Data.
        END IF

                GOSUB WriteToFile
                ON TIMER(1,200) GOSUB ClearLCD                  ' TIMER(1): 2 sec
                ScanFlag% = 0

Exit_ReadData:
        ON TIMER(2,60000) GOSUB ForceToLogout                   ' TIMER(2): 10 min
        RETURN

'*************************************************************************************************
' Routine: WriteToFile                                                                           *
' Purpose:To record the User ID, date, time, ErrorFlag and the scanned label to DBF2.            *
' Return:                                                                                        *
' Call:                                                                                          *
'*************************************************************************************************
WriteToFile:
        Data$ = DATE$ + TIME$ + UserID$ + ErrorIndex$ + Data$
        ADD_RECORD(2,Data$)
        RETURN

'*************************************************************************************************
' Routine: ClearLCD                                                                              *
' Purpose:To clear the data which have been shown on the LCD for the duration defined            *
'         by timer(1).                                                                           *
' Return:                                                                                        *
' Call:                                                                                          *
'*************************************************************************************************
ClearLCD:
        OFF TIMER(1)
        CLS
        RETURN


'### end of program ###
```

Appendix A: Version History

(1) Ver. 1.00, Nov 1997
- First release.

(2) Ver. 1.10, Jun 09 1998
- Modified: Selection of “Target Machine” is canceled. Target machine is automatically determined by the Run-time Engine.
- New: The On-line Help is added.

(3) Ver. 1.22, Dec 01 1998
- New: The “RETURN” of a subroutine can be appended with a label.

(4) Ver. 1.24, Feb 26, 1998
- New: CURSOR (N%)
- New: ON ESC GOSUB *Subname*
- New: OFF ESC

(5) Ver. 1.26, Mar 22, 1999
- Modified: UPCE no addon => UPCE
- Modified: EAN 8 no addon => EAN 8
- Modified: EAN 13 no addon => EAN 13 & UPCA + Config.

(6) Ver. 1.27, Mar 30, 1999
- New: GET_TRANSACTION_DATA$(N%)
- New: DEL_TRANSACTION_DATA(N%)
- New: SET_VEDIO_MODE(N%)
- New: SHOW_IMAGE(x%, y%,size_x%, size_y%, Image$)
- New: SELECT_FONT(N%)
- New: CHANGE_SPEED(N%)

(7) Ver. 1.28, Apr 20, 1999
- New: Download via Serial IR Transceiver (for 7xx terminals)

(8) Ver. 1.29, Apr 30, 1999
- New: COM_DELIMITER(N%, C%)

(9) Ver. 1.30, Jun 01, 1999
- New: DEVICE_ID$
- New: MAIN_BATTERY
- New: BACKUP_BATTERY
- New: RAM_SIZE
- New: ROM_SIZE
- New: FREE_MEMORY
- New: CODE_TYPE
- New: DAY_OF_WEEK

(10) Ver. 1.31, Jul 14, 1999
- Modified: If no index file is created for a DBF file, the related settings such as Offset and Key Length will be set to 0.

(11) Ver. 1.32, Aug 14, 1999
- New: SET_COM_TYPE(ComPort%, Type%)

(12)Ver. 1.33, Sep 07, 1999

- New: SET_WEDGE(Parameter$)
- New: SEND_WEDGE(Data$)

(13)Ver. 1.34, Oct 15, 1999

- New: Selection of Target Machine is added.
- New: Communication baud rate is up to 115200 bps for 520 and 7xx terminals.

(14)Ver. 1.40, Nov 02, 1999

- Modified: Supports up to 5 DBF files, and up to 3 IDX files for each DBF file.

(15)Ver. 1.50, Dec 03, 1999

- Modified: Supports up to 6 transaction files for 7xx terminals.
- New: UPDATE_RECORD_EX(File%, Index%, Data$)
- New: UPDATE_TRANSACTION_EX(File%, Record%, Data$)
- New: SAVE_TRANSACTION_EX(File%, Data$)
- New: GET_TRANSACTION_DATA_EX(File%, Record%)
- New: EMPTY_TRANSACTION_EX(File%)
- New: DEL_TRANSACTION_DATA_EX(File%, Record%)
- New: TRANSACTION_COUNT_EX(File%)

(16)Ver. 1.53, Jan 07, 2000

- New: Supports new scan mode – “Aiming Mode”.

(17)Ver. 1.54, Jan 14, 2000

- New: SMC_SIZE
- New: SMC_FREE_MEMORY

(18)Ver. 1.55, Feb 01, 2000

- Modified: A compilation bug is fixed.

(19)Ver. 1.60, Feb 18, 2000

- New: ALPHA_LOCK
- New: GET_ALPHA_LOCK
- New: SHIFT_LOCK
- New: GET_SHIFT_LOCK
- New: KEY_CLICK
- New: FILL_RECT
- New: CLR_RECT
- New: VERSION

(20)Ver. 2.00, Apr 20, 2000

- Modified: On-line Help

Appendix B: Flash Writer

The 201/510 Flash Writer is an extension board to provide the download function when the existing program does not support download function and/ or the existing program is corrupted. A step-by-step operation procedure is listed as follows.

(1) Turn OFF the power of the terminal.

(2) Remove the flash memory from the terminal PCB.

(3) Connect the end of the flat cable to the flash memory socket on the terminal PCB. The RED wire should connect to pin 1 of the socket.

(4) Insert the flash memory into socket U2. Press the handle to hold it tight.

(5) There is a single wire with hooker on the flash writer. Connect the hooker to any pin of JP7 (for 201), or JP1 (for 510).

(6) Connect the RS-232 cable to PC. For 201, use COM1 (C2) or COM3 (Aux 232). For 510, use COM1 (phone-jack) or COM3 (D-25P)

(7) Make sure that all connections have been correctly connected.

(8) Turn on the flash writer, the yellow LED will light.

(9) A short beep will be activated if everything is working fine and the display will show

#3 02032007 00001

where,

- #N: the located flash address which depends on the pin connected,
  for 201 : 1 to 3
  for 510 : 4 to 7
- 02032007: first 4 digits can be ignored (similar to N as just described). Last 4 digits show the type of the flash memory.
  2007 – SGS Thompson 28F101
  01A7 – AMD 28F010
  ABCD – AMD 29F010
  89B4 – Intel 28F010
- 00001: 5 digits system second counts, which would be updated periodically. An easy way to see if the system is working properly.

(10) The RS-232 port has been fixed to 38400, N, 8

(11) Download program code from PC.

(12) After downloading is completed, turn off the terminal.

(13) Straight-up the handle and remove the flash chip from the socket.

(14) Put the flash chip back into the terminal.

(15) Power on the terminal to start the new program.

Appendix C: Run-Time Error Table

Error Code	Explanation
1	Unknown operator
2	Operand count mismatch
3	Type mismatch
4	Can’t perform type conversion
5	No available temp string
6	Illegal operand
7	Not an L-value
8	Float error
9	Bad array subscript
10	Unknown function
11	Illegal function call
12	Return without GOSUB

Appendix D: Barcode Setting

The CipherLab terminals support the decodability of several barcode symbologies, including Code 39, Italy phamacode, French phamacode, Industrial 2 of 5, Interleave 2 of 5, Matrix 2 of 5, Codabar, MSI, Plessey, Code 93, UPCE, UPCE w/ Addon 2, UPCE w/ Addon 5, EAN 8, EAN 8 w/ Addon 2, EAN 8 w/ Addon 5, EAN 13 & UPCA, EAN 13 w/ Addon 2, EAN 13 w/ Addon 5, Code 128, and so on.

The BASIC Compiler provides the user a menu-driven interface to configure the decodability of these barcode symbologies and also the scanner behavior. On the “Barcode Setting” window, the user can check the box in front of a barcode type to enable the decodability with respect to the barcode type. For some of the supported barcode symbologies, the user may click the “Config” button to do more configurations, such as enable/disable the checksum verification.

This appendix describes these symbology parameters and scanner parameters.

D.1. Symbology Parameters

D.1.1. Code 39

- Transmit Start/Stop

  This parameter specifies whether the start/stop characters of Code 39 are included in the data being transmitted.

- Verify Checksum

  If this parameter is enabled, the target terminal will perform checksum verification when decoding Code 39 barcodes. If the checksum is incorrect, the barcode will not be accepted.

- Transmit Checksum

  If this parameter is enabled, the checksum character will be included in the data being transmitted.

- Code 39 Full ASCII

  User can check the box to read Full ASCII Code 39.

D.1.2. Italy / French Phamacode

- Transmit Checksum

  If this parameter is enabled, the checksum character will be included in the data being transmitted.

D.1.3. Industrial / Interleave / Matrix 2 of 5

- Start/Stop Selection

  This parameter provides the readability of all 2 of 5 symbology variants. For example, flight tickets actually use an Industrial 2 of 5 barcode but with Interleave 2 of 5 start/stop. In order to read this barcode, the start/stop selection parameter of Industrial 2 of 5 should set to “Interleave 25”.

- Verify Checksum

  If this parameter is enabled, the target terminal will perform checksum verification when decoding these barcodes. If the checksum is incorrect, the barcode will not be read.

- Transmit Checksum

  If this parameter is enabled, the checksum character will be included in the data being transmitted.

- Length Qualification

  Because of the weak structure of the 2 of 5 barcodes, it is possible to make a “short scan” error. To prevent the “short scan” error, user can define the “Length Qualification” settings to insure that the correct code is read by qualifying the allowable code length. The barcode can be qualified by “Fixed Length” or “Max/Min Length”. If “Fixed Length” is selected, up to 2 fixed lengths can be specified. If “Max/Min Length” is selected, the maximum length and the minimum length must be specified. The target terminal will only accept those barcodes with lengths that fall between max/min lengths specified.

D.1.4. Codabar

- Start/Stop Character

  User can select no start/stop characters or one of the four different start/stop character pairs, i.e., abcd/abcd, abcd/tn*e, ABCD/ABCD, and ABCD/TN*E, to be included in the data being transmitted.

D.1.5. MSI

- Checksum Verification

  User can select one of the three kinds of checksum calculations, i.e., Single Modulo 10, Double Modulo 10, and Modulo 11 & 10, to verify MSI code. If the checksum character is incorrect, the barcode will not be read.

- Checksum Transmission

  This parameter specifies how the checksum is to be transmitted. User can select from “Last digit not transmitted”, “Transmitted”, and “Last 2 digits not transmitted”.

- Length Qualification

  Because of the weak structure of the MSI code, it is possible to make a “short scan” error. To prevent the “short scan” error, user can define the “Length Qualification” settings to insure that the correct code is read by qualifying the allowable code length. The barcode can be qualified by “Fixed Length” or “Max/Min Length”. If “Fixed Length” is selected, up to 2 fixed lengths can be specified. If “Max/Min Length” is selected, the maximum length and the minimum length must be specified. The target terminal will only accept MSI code with lengths that fall between max/min lengths specified.

D.1.6. Plessey

- Convert to UK Plessey

  If this parameter is enabled, the target terminal will change each occurrence of the character ‘A’ to character ‘X’ in the code.

- Checksum Transmission

  If this parameter is enabled, the checksum characters (two characters) will be transmitted together with data.

D.1.7. UPCE

- Convert to UPCA

  If this parameter is enabled, the read UPCE barcode will be expanded into UPCA, and the next processing will follow the parameters configured for UPCA.

- Transmit System Number

  If this parameter is enabled, the system number will be included in the data being transmitted.

- Transmit Checksum

  If this parameter is enabled, the checksum character will be included in the data being transmitted.

D.1.8. EAN8

- Convert to EAN 13

  If this parameter is enabled, the EAN 8 read will be expanded into EAN 13, and the next processing will follow the parameters configured for EAN 13.

- Transmit Checksum

  If this parameter is enabled, the checksum character will be included in the data being transmitted.

D.1.9. EAN13 & UPCA

- ISBN Conversion

  If this parameter is enabled, the EAN 13 codes starting with 978 and 979 will be converted to ISBN code.

- ISSN Conversion

  If this parameter is enabled, the EAN 13 codes starting with 977 will be converted to ISSN code.

- Transmit Checksum

  If this parameter is enabled, the EAN 13 checksum character will be included in the data being transmitted.

- Transmit UPCA System Number

  If this parameter is enabled, the UPCA system number will be included in the data being transmitted.

- Transmit UPCA Checksum

  If this parameter is enabled, the UPCA checksum character will be included in the data being transmitted.

D.2. Scanner Parameters

The user can define the scanner parameters for reader ports 1 and 2. Since the 7xx terminals support only one reader port, the settings for reader port 2 will be ignored if the target is a 7xx terminal.

D.2.1. Scan Mode

- Auto Off Mode

  The scanner will start scanning once the switch is triggered. The scanning continues until either a barcode is read or a preset scanning period (Time-Out) has expired. This is the default scan mode.

- Continuous Mode

  The scanner is always scanning but will just decode the same barcode once. To read the same barcode, the barcode must be taken away from the scanning line and back again.

- Auto Power Off Mode

  The scanner will start scanning once the switch is triggered. The scanning continues until a preset scanning Time-Out period is expired. Unlike the Auto Off mode, the scanner will continue to scan and the scanning period is re-counted whenever there is a successful read.

- Alternate Mode

  The scanner will start scanning once the switch is triggered. The scanner will continue scanning until the switch is triggered again.

- Momentary Mode

  The scanner will be scanning as long as the switch is pressed.

- Repeat Mode

  The scanner is always scanning just like Continuous Mode. But in this mode the switch acts like a “re-transmit button”. If the switch is triggered within 1 second after a good read, the same data will be transmitted again without actually reading the barcode. This “re-transmit button” can be triggered as many times as the user desires, as long as the time between each triggering does not exceed 1 second. This scan mode is most useful when the same barcode is to be read many times.

- Laser Mode

  This is the scan mode used most often on laser scanners. The scanner will start scanning once the switch is pressed. The scanning goes on until either a barcode is read or the switch is released.

- Test Mode

  The scanner is always scanning and will decode repeatedly even with the same barcode.

- Aiming Mode

  By selecting this mode, the user needs to trigger twice to decode. That is, the first trigger is for aiming only, while the second trigger will truly start to decode. After first trigger, the scanner will keep on scanning for one second so that the user may take aim. But the user must press the second trigger within this period (default of one second), otherwise it will be reset and the user has to take aim again. This mode is used when two barcodes are printed too close together so that users ensure they read the correct barcode.

D.2.2. Read Redundancy

This parameter is used to specify the level of reading security. If “No Redundancy” is selected, one successfully decoded barcode will make the reading valid and induce the “READER Event”. If “Three Times” is selected, it will take four consecutive successful decodes of the same barcode to make the reading valid. The more redundancy the user selects, the higher the reading security but the slower the reading speed. The user must compromise between reading security and decoding speed.

D.2.3. Time-Out

This parameter is the scanning period, between 0 to 255 seconds, for Auto Off Mode and Auto Power Off Mode. If the scanning period is expired, the scanner will stop scanning.

D.2.4. Negative Barcode

Normally, barcodes are printed with the color of the bars darker than that of the spaces. But for negative barcodes, they are printed in the opposite sense, just like negative films. The spaces of the negative barcodes are printed with a color darker than that of the bars. User can check the box of “Read Negative Barcode” to enable the readability of negative barcodes.

Appendix E: Pseudo Keyboard

Pseudo keyboard is not a real keyboard, but a special set of barcodes. When the pseudo keyboard is enabled, the user can scan the barcodes of the pseudo keyboard instead of making keystrokes on the real keyboard. The BASIC Run-Time Engines will interpret a scan of the barcode "~x" to the keystroke "x", where "x" is a printable character (ASCII 20 – 126), the barcode "\A" – "\L" to F1 – F12, the barcode "\M" to "Enter", the barcode "\N" to "Back Space", and the barcode "\O" to "Esc". Please note that all the barcodes need to be printed in Code128. A set of the pseudo keyboard is printed as follows.

F1

F2

F3

F4

F5

F6

F7

F8

F9

F10

F11

F12

Enter

Backspace

Esc

Space

!

"

#

$

%

&

'

(

)

*****

+

,

-

.

/

0

1

2

3

4

5

6

7

8

9

:

;

<

=

>

?

@

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

[

\

]

^

_

`

a

b

c

d

e

f

g

h

i

j

k

l

m

n

o

p

q

r

s

t

u

v

w

x

y

z

{

|

}

~

Appendix F: Debug Messages

Debug messages indicate the activities happening on the system. The common debug messages are listed as follows.

Message	Explanation
ABS(*N*%)	Indicating the command ABS is processed.
ADD(*N1%*,*N2%*)	Indicating an addition is processed.
ADD_RECORD(*N*%, *A$*)	Indicating the command ADD_RECORD is processed. N% is the number of the DBF file. A$ is the data to be added to the DBF file.
ALPHA_LOCK(*N*%)	Indicating the command ALPHA_LOCK is processed. N% is the Alpha Lock status.
AND	Indicating the logical operation AND is processed.
ARY(*N%*)	Indicating an N-element array is declared.
ASC(*A*$)	Indicating the command ASC is processed.
ASGN(*A*)	Indicating that the value A is assigned to the variable. A could be an integer, long integer, character, string, or any type.
BACKLIT(N%)	Indicating the command BACKLIT is processed. N% is the backlight status
BACKUP_BATTERY	Indicating the command BACKUP_BATTERY is processed.
BEEP(…)	Indicating the command BEEP is processed.
CAP_LOCK(*N*%)	Indicating the command CAP_LOCK is processed. N% is the Cap Lock status.
CHANGE_SPEED(*N*%)	Indicating the command CHANGE_SPEED is processed. N% is the selection of the speed.
CHR$(*N*%)	Indicating the command CHR is processed.
CLOSE_COM(*N*%)	Indicating the command CLOSE_COM is processed. N% is the number of the COM port.
CLR_RECT(…)	Indicating the command CLR_RECT is processed.
CLS	Indicating the command CLS is processed.
CODE_TYPE	Indicating the command CODE_TYPE is processed.
CURSORX	Indicating the command CURSOR_X is processed.
CURSORY	Indicating the command CURSOR_Y is processed.
DATE$	Indicating the system date is inquired.
DATE$(*A$*)	Indicating the system date is updated. A$ is the new system date.
DAY_OF_WEEK	Indicating the command DAY_OF_WEEK is processed.
DEL_RECORD(*N1%*,*N2%*)	Indicating the command DEL_RECORD is processed. N1% is the number of the DBF file; N2% is the number of the IDX file.
DEL_TRANSACTION_DATA	Indicating the command DEL_TRANSACTION_DATA is processed.
DEL_TRANSACTION_DATA_EX(*N*%)	Indicating the command DEL_TRANSACTION_DATA_EX is processed. N% is the number of the transaction file.
DIGOUT(*N1%*,*N2%*,*N3%*)	Indicating the command DIGOUT is processed.
DISABLE_KBD	Indicating the command DISABLE KBD is processed.
DISABLE_PSEUDO_KBD	Indicating the command DISABLE PSEUDO_KBD is processed.
DISABLE_READER(*N*%)	Indicating the command DISABLE READER is processed. N% is the number of the reader port.
DIV(*N1%*,*N2%*)	Indicating a division is processed.
EMPTY_FILE(*N*%)	Indicating the command EMPTY_FILE is processed. N% is the number of the DBF file.
EMPTY_TRANSACTION	Indicating the command EMPTY_TRANSACTION is processed.
EMPTY_TRANSACTION_EX(*N*%)	Indicating the command EMPTY_TRANSACTION_EX is processed. N% is the number of the transaction file.

ENABLE_KBD	Indicating the command ENABLE KBD is processed.
ENABLE_PSEUDO_KBD	Indicating the command ENABLE PSEUDO_KBD is processed.
ENABLE_READER(*N*%)	Indicating the command ENABLE READER is processed. N% is the number of the reader port.
EQU? (*N1*%,*N2*%)	Indicating the decision “IF N1% = N2%” is processed.
EVENT(0)	Indicating the “COM(1) EVENT” happens.
EVENT(1)	Indicating the “COM(2) EVENT” happens.
EVENT(2)	Indicating the “COM(3) EVENT” happens.
EVENT(3)	Indicating the “INQUIRY EVENT” happens.
EVENT(4)	Indicating the “DIGIN(1) EVENT” happens.
EVENT(5)	Indicating the “DIGIN(2) EVENT” happens.
EVENT(6)	Indicating the “DIGIN(3) EVENT” happens.
EVENT(7)	Indicating the “DIGIN(4) EVENT” happens.
EVENT(8)	reserved
EVENT(9)	Indicating the “TIMER(1) EVENT” happens.
EVENT(10)	Indicating the “TIMER(2) EVENT” happens.
EVENT(11)	Indicating the “TIMER(3) EVENT” happens.
EVENT(12)	Indicating the “TIMER(4) EVENT” happens.
EVENT(13)	Indicating the “TIMER(5) EVENT” happens.
EVENT(14)	Indicating the “ON MINUTE EVENT” happens.
EVENT(15)	Indicating the “ON HOUR EVENT” happens.
EVENT(16)	Indicating the “READER(1) EVENT” happens.
EVENT(17)	Indicating the “READER(2) EVENT” happens.
EVENT(18)	Indicating the “FUNCTION(1) EVENT” happens.
EVENT(19)	Indicating the “FUNCTION(2) EVENT” happens.
EVENT(20)	Indicating the “FUNCTION(3) EVENT” happens.
EVENT(21)	Indicating the “FUNCTION(4) EVENT” happens.
EVENT(22)	Indicating the “FUNCTION(5) EVENT” happens.
EVENT(23)	Indicating the “FUNCTION(6) EVENT” happens.
EVENT(24)	Indicating the “FUNCTION(7) EVENT” happens.
EVENT(25)	Indicating the “FUNCTION(8) EVENT” happens.
EVENT(26)	Indicating the “FUNCTION(9) EVENT” happens.
EVENT(27)	Indicating the “FUNCTION(10) EVENT” happens.
EVENT(28)	Indicating the “FUNCTION(11) EVENT” happens.
EVENT(29)	Indicating the “FUNCTION(12) EVENT” happens.
EVENT(30)	reserved
EVENT(31)	Indicating the “ESC EVENT” happens.
EXP(*N1*%,*N2*%)	Indicating an exponentiation is processed.
FALSE?(*N*%)	Indicating the “IF” statement or the “WHILE” statement is processed.
FILL_RECT(…)	Indicating the command FILL_RECT is processed.
FIND_RECORD(*N1*%,*N2*%,*A$*)	Indicating the command FIND_RECORD is processed. N1% is the number of the DBF file; N2% is the number of the IDX file; A$ is the key string to be found.
FREE_MEMORY	Indicating the command FREE_MEMORY is processed.

GET_ALPHA_LOCK	Indicating the command GET_ALPHA_LOCK is processed.
GET_CTS(*N*%)	Indicating the command GET_CTS is processed. N% is the number of the COM port.
GET_DEVICE_ID	Indicating the command GET_DEVICE_ID is processed.
GET_DIGIN(*N*%)	Indicating the command GET_DIGIN is processed. N% is the number of the COM port.
GET_FILE_ERROR	Indicating the command GET_FILE_ERROR is processed.
GET_INQUIRY$	Indicating the command GET_INQUIRY$ is processed.
GET_READER_DATA$(*N*%)	Indicating the command GET_READER_DATA$ is processed. N% is the number of the reader port.
GET_READER_SETTING(*N*%)	Indicating the command GET_READER_SETTING is processed. N% is the setting number.
GET_RECORD$(*N1*%,*N2*%)	Indicating the command GET_RECORD$ is processed. N1% is the number of the DBF file; N2% is the number of the IDX file.
GET_RECORD_NUMBER(*N1*%,*N2*%)	Indicating the command GET_READER_NUMBER is processed. N1% is the number of the DBF file; N2% is the number of the IDX file.
GET_SHIFT_LOCK	Indicating the command GET_SHIFT_LOCK is processed.
GET_STATION_ID	Indicating the command GET_STATION_ID is processed.
GET_TRANSACTION_DATA$(*N*%)	Indicating the command GET_TRANSACTION_DATA is processed. N% is the ordinal number of the record to be read.
GET_TRANSACTION_DATA_EX$(*N1*%, *N2*%)	Indicating the command GET_TRANSACTION_DATA_EX is processed. N1% is the number of the transactin file, N2% is the ordinal number of the record to be read.
GOSUB(*N*%)	Indicating the program branches to a subroutine. N% is the line number of the first line of the subroutine.
GOTO(*N*%)	Indicating the program branches to line number N%.
GT? (*N1*%,*N2*%)	Indicating the decision “IF N1% > N2%” is processed.
HEX$(*N*%)	Indicating the command HEX$ is processed.
HIDE_CALENDAR	Indicating the command HIDE_CALENDAR is processed.
INKEY$(*A$*)	Indicating the command INKEY is processed.
INPUT	Indicating the command INOUT is processed.
INPUT_MODE(*N*%)	Indicating the command INPUT_MODE is processed. N% is the setting for the input mode.
INQUIRY$(*A$,N*%)	Indicating the command INQUIRY$ is processed. A$ is the inquiry message; N% is the setting for the timeout.
INSTR(*N*%,*A1*$,*A2*$)	Indicating the command INSTR is processed.
INT(*N*%)	Indicating the command INT is processed.
KEY_CLICK(*N*%)	Indicating the command KEY_CLICK is processed. N% is the setting for the key click sound.
L(*N*%)	Indicating the line number being executed
LCASE$(*A*$)	Indicating the command LCASE$ is processed.
LE? (*N1*%,*N2*%)	Indicating the decision “IF N1% <= N2%” is processed.
LED(*N1*%,*N2*%,*N3*%)	Indicating the command LED is processed. N1%, N2%, N3% are the LED settings.
LEFT$(*A*$)	Indicating the command LEFT$ is processed.
LEN(*A*$)	Indicating the command LEN is processed.
LOCATE(*N1*%,*N2*%)	Indicating the command LOCATE is processed.
LOCK	Indicating the command LOCK is processed.
LT? (*N1*%,*N2*%)	Indicating the decision “IF N1% < N2%” is processed.
MAIN_BATTERY	Indicating the command MAIN_BATTERY is processed.
MID$(*A*$)	Indicating the command MID$ is processed.
MOD(*N1*%,*N2*%)	Indicating a modulo operation is processed.

MOVE_TO(*N1*%,*N2*%,*N3*%)	Indicating the command MOVE_TO is processed. N1% is the number of the DBF file; N2% is the number of the IDX file; N3% is the record number to move to.
MOVE_TO_NEXT(*N1*%,*N2*%)	Indicating the command MOVE_TO_NEXT is processed. N1% is the number of the DBF file; N2% is the number of the IDX file.
MOVE_TO_PREVIOUS(*N1*%,*N2*%)	Indicating the command MOVE_TO_PREVIOUS is processed. N1% is the number of the DBF file; N2% is the number of the IDX file.
MUL(*N1*%,*N2*%)	Indicating a multiplication is processed.
NEG (*N1*%)	Indicating a negation is processed.
NEQ? (*N1*%,*N2*%)	Indicating the decision “IF N1% <> N2%” is processed.
NOT	Indicating the logical operation NOT is processed.
NUM_LOCK(*N*%)	Indicating the command NUM_LOCK is processed. N% is the Num Lock status.
OCT$(*A*$)	Indicating the command OCT$ is processed.
OFF_ALL	Indicating the command OFF ALL is processed.
OFF_COM(*N*%)	Indicating the command OFF COM is processed. N% is the number of the COM port.
OFF_DIGIN(*N*%)	Indicating the command OFF DIGIN is processed. N% is the number of the digital input port.
OFF_ENQUIRY	Indicating the command OFF ENQUIRY is processed.
OFF_ESC	Indicating the command OFF ESC is processed.
OFF_HOUR_SHARP	Indicating the command OFF HOUR_SHARP is processed.
OFF_KEY	Indicating the command OFF KEY is processed.
OFF_MINUTE_SHARP	Indicating the command OFF MINUTE_SHARP is processed.
OFF_NET	Indicating the command OFF NET is processed.
OFF_READER(*N*%)	Indicating the command OFF READER is processed. N% is the number of the reader port.
OFF_TIMER(*N*%)	Indicating the command OFF TIMER is processed. N% is the number of the timer.
ON_COM(*N1%, N2*%)	Indicating the command ON COM GOSUB is called. N1% is the umber of the COM port; N2% is the line number of the subroutine to branch to.
ON_DIGIN(*N1%, N2*%)	Indicating the command ON DIGIN GOSUB is called. N1% is the line number of the digital input port; N2% is the subroutine to branch to.
ON_ENQUIRY(*N*%)	Indicating the command ON ENQUIRY GOSUB is called. N% is the line number of the subroutine to branch to.
ON_ESC(*N*%)	Indicating the command ON ESC GOSUB is called. N% is the line number of the subroutine to branch to.
ON_GOSUB(*N*%)	Indicating the command ON GOSUB is called. N% is the line number of the subroutine to branch to.
ON_GOTO(*N*%)	Indicating the command ON GOTO is called. N% is the line number of the subroutine to branch to.
ON_HOUR_SHARP(*N*%)	Indicating the command ON HOUR_SHARP GOSUB is called. N% is the line number of the subroutine to branch to.
ON_KEY(*N*%)	Indicating the command ON KEY GOSUB is called. N% is the line number of the subroutine to branch to.
ON_MINUTE_SHARP(*N*%)	Indicating the command ON MINUTE_SHARP GOSUB is called. N% is the line number of the subroutine to branch to.
ON_NET(*N*%)	Indicating the command ON NET GOSUB is called. N% is the line number of the subroutine to branch to.
ON_READER(*N1*%,*N2*%)	Indicating the command ON READER GOSUB is called. N1% is the number of the reader port; N2% is the line number of the subroutine to branch to.
ON_TIMER(*N1*%,*N2*%)	Indicating the command ON TIMER GOSUB is called. N1% is the number of the timer; N2% is the line number of the subroutine to branch to.
OPEN_COM(*N1%*)	Indicating the command OPEN_COM is processed. N1% is the number of the COM port.
OR	Indicating the logical operation OR is processed.
PRINT(*A*$)	Indicating the command PRINT is processed.

RAM_SIZE	Indicating the command RAM_SIZE is processed.
READ_COM$(*N*%)	Indicating the command READ_COM$ is processed. N% is the number of the COM port.
READ_NET$	Indicating the command READ_NET$ is processed.
READER_SETTING(*N1*%,*N2*%)	Indicating the command READER_SETTING is processed. N1% is the setting number; N2% is the value of the setting.
RECORD_COUNT(*N*%)	Indicating the command RECORD_COUNT is processed. N% is the number of the DBF file.
REPLY(*A*$)	Indicating the command REPLY is processed. A$ the message to reply.
RETURN(*N*%)	Indicating the command RETURN is processed. N% is the line number to return, if it is not null.
RIGHT$(*A*$)	Indicating the command RIGHT$ is processed.
ROM_SIZE	Indicating the command ROM_SIZE is processed.
SAVE_TRANSACTION(*A*$)	Indicating the command SAVE_TRANSACTION. A$ is the data to be saved.
SAVE_TRANSACTION_EX(*N*%,*A*$)	Indicating the command SAVE_TRANSACTION_EX. N% is the number of the transaction file; A$ is the data to be saved.
SELECT_FONT(*N*%)	Indicating the command SELECT_FONT is processed. N% is the selection of the font size.
SEND_WEDGE(*A*$)	Indicating the command SEND_WEDGE is processed. A$ is the character string be sent.
SET_COM(*N1%,N2%,N3%,N4%,N5%*)	Indicating the command SET_COM is processed. N1% is the number of the COM port; N2%, N3%, N4%, N5% are the settings of the COM port.
SET_COMM_TYPE(*N1*%,*N2*%)	Indicating the command SET_COMM_TYPE is processed. N1% is the number of the COM port; N2% is the type of the COM port.
SET_CURSOR(*N*%)	Indicating the command SET_CURSOR is processed. N% is the cursor status.
SET_PRECISION(*N*%)	Indicating the command SET_PRECISION is processed. N% is the numerical precision.
SET_RTS(*N1*%,*N2*%)	Indicating the command SET_RTS is processed. N1% is the number of the COM port; N2% is the RTS status.
SET_STATION_ID(*N*%)	Indicating the command SET_STATION_ID is processed. N% is the new station ID.
SET_VIDEO_MODE(*N*%)	Indicating the command SET_VIDEO_MODE is processed. N% is the video mode selected.
SET_WEDGE(*A*$)	Indicating the command SET_WEDGE is processed. *A*$ is the character array for the Wedge Settings.
SHIFT_LOCK(*N*%)	Indicating the command SHIFT_LOCK is processed. N% is the shift lock status.
SHOW_CALENDAR(*N1%,N2%,N3*%)	Indicating the command SHOW_CALENDAR is processed.
SHOW_IMAGE(…)	Indicating the command SHOW_IMAGE is processed.
SIGN(*N*%)	Indicating the command SGN is processed.
SMC_FREE_MEMORY	Indicating the command SMC_FREE_MEMORY is processed.
SMC_SIZE	Indicating the command SMC_SIZE is processed.
START_NETWORK	Indicating the command START NETWORK is processed.
STOP	Indicating the command STOP is processed.
STOP_BEEP	Indicating the command STOP BEEP is processed.
STOP_NETWORK	Indicating the command STOP NETWORK is processed.
STR$(*N*%)	Indicating the command STR$ is processed.
STRING$(*N1*%, *N2*%)	Indicating the command STRING$ is processed.
SUB(*N1*%, *N2*%)	Indicating a subtraction is processed.
T(*N*%)	Indicating the stack's level. When the program branches to a subroutine, the stack's level increases 1; when the program returns, the stack's level decreases 1. It can be used to check if the "stack overflow" problem happens.
TIME$	Indicating the system time is inquired.

TIME$(*A*$)	Indicating the system time is updated. A$ is the new system time.
TIMER	Indicating the command TIMER is processed.
TRANSACTION_COUNT	Indicating the command TRANSACTION_COUNT is processed.
TRANSACTION_COUNT_EX(*N*%)	Indicating the command TRANSACTION_COUNT_EX is processed. N% is the number of the transaction file.
TRIM_LEFT$(*A*$)	Indicating the command TRIM_LEFT$ is processed.
TRIM_RIGHT$(*A*$)	Indicating the command TRIM_RIGHT$ is processed.
UCASE$(*A*$)	Indicating the command UCASE$ is processed.
UNLOCK	Indicating the command UNLOCK is processed.
UPDATE_RECORD_EX(*N1*%,*N2*%,*A$*)	Indicating the command UPDATE_RECORD_EX is processed. N1% is the number of the DBF file; N2% is the number of the IDX file; A$ is the new data.
UPDATE_TRANSACTION_EX(*N*%,*A$*)	Indicating the command UPDATE_TRANSACTION_EX is processed. N% is the number of the transaction file; A$ is the new data.
VAL(*A*$)	Indicating the command VAL is processed.
VERSION(*A*$)	Indicating the command VERSION is processed. A$ is the character string to be written as the version information.
VOL(*N*%)	Indicating the command VOL is processed.
WAIT(*N*%)	Indicating the command WAIT is processed.
WRITE_COM(*N*%,*A*$)	Indicating the command WRITE_COM is processed.
WRITE_NET->	Indicating the command WRITE_NET is processed.
XOR	Indicating the logical operation XOR is processed.

Index

A

ABS, 21
ADD_RECORD, 89
ALPHA_LOCK, 66
arithmetic operator, 14
array, 13
ASC, 32
assignment operator, 14

B

BACKLIT, 70
BACKUP_BATTERY, 75
Barcode Setting, 10, 123
  2 of 5 Code, 123
  Codabar, 124
  Code 39, 123
  EAN 13, 125
  EAN 8, 125
  MSI, 124
  Phamacode, 123
  Plessey, 124
  UPCA, 125
  UPCE, 124
Battery, 75
BEEP, 58
Buzzer, 58

C

Calendar, 60
CAP_LOCK, 66
CHANGE_SPEED, 45
CHR$, 32
CLOSE_COM, 76
CLR_KBD, 66
CLR_RECT, 70
CLS, 71
CODE_TYPE, 47
COM_DELIMITER, 76
Communication Ports, 76
compile, 10
constant, 12
CPU running speed, 45
CURSOR, 71
CURSOR_X, 71
CURSOR_Y, 72

D

DAT, 84
DATE$, 60
DAY_OF_WEEK, 60
DBF, 10, 89
Debug, 97
  Debug Message, 130
Decision Structures, 24
DEL_RECORD, 90
DEL_TRANSACTION_DATA, 84
DEL_TRANSACTION_DATA_EX, 84
DEVICE_ID$, 45
Digital I/O, 63
DIGOUT, 63
DIM, 21
DISABLE KBD, 66
DISABLE PSEUDO_KBD, 67
DISABLE READER, 47
download, 10

E

EMPTY_FILE, 90
EMPTY_TRANSACTION, 85
EMPTY_TRANSACTION_EX, 85
ENABLE KBD, 67
ENABLE PSEUDO_KBD, 67
ENABLE READER, 48
Error Code, 95
Event Triggers, 35
EXIT, 27

F

File Manipulation, 84
FILL_RECT, 72
FIND_RECORD, 91
Flash Writer, 121
Font Files, 70
FOR… NEXT, 27
FREE_MOMERY, 96

G

GET_ALPHA_LOCK, 68
GET_CTS, 77
GET_DIGIN, 63
GET_FILE_ERROR, 95
GET_INQUIRY$, 81
GET_READER_DATA$, 48
GET_READER_SETTING, 48
GET_RECORD, 91
GET_RECORD_NUMBER, 92
GET_SHIFT_LOCK, 68
GET_STATION_ID, 45
GET_TRANSACTION_DATA$, 86
GET_TRANSACTION_DATA_EX$, 86
GOTO, 21

H

HEX$, 32
HIDE_CALENDAR, 61
Host Commands, 97
  CLEAR, 97
  READ, 97
  REMOVE, 98
  TR, 98
  TW, 98
  X, 99

I

IDX, 89
IF… THEN… ELSE, 24
IF… THEN… END IF, 26
INKET$, 68
INPUT, 69
INPUT_MODE, 69
INQUIRY$, 81
INSTR, 30
INT, 22

K

KEY_CLICK, 58
Keyboard, 66
Keyboard Wedge, 53

L

label, 17
LCASE$, 32
LCD, 70
LED, 64
LEFT$, 30
LEN, 29
LOCATE, 72
LOCK, 44
logical operator, 15
Looping Structures, 27

M

MAIN_BATTERY, 75
Memory, 96
MID$, 30
MOVE_TO, 92
MOVE_TO_NEXT, 93
MOVE_TO_PREVIOUS, 93

N

Negative Barcode, 126
NUM_LOCK, 69
numeric, 12

O

OCT$, 33
OFF ALL, 36
OFF COM, 36
OFF DIGIN, 36
OFF ENQUIRY, 37
OFF ESC, 37
OFF HOUR_SHARP, 37
OFF KEY, 38
OFF MINUTE_SHARP, 38
OFF NET, 38
OFF READER, 39
OFF TIMER, 39
ON COM GOSUB, 39
ON DIGIN GOSUB, 40
ON ENQUIRY GOSUB, 41
ON ESC GOSUB, 40
ON HOUR_SHARP GOSUB, 41
ON KEY GOSUB, 41
ON MINUTE_SHARP GOSUB, 42
ON NET GOSUB, 42
ON READER GOSUB, 43
ON TIMER GOSUB, 43
ON… GOSUB, 25
ON… GOTO, 25
OPEN_COM, 77
operator precedence, 16

P

PRINT, 72
Pseudo Keyboard, 127

R

Read Redunancy, 126
READ_COM$, 77, 80
READ_NET$, 82
Reader, 47
READER_SETTING, 49
RECORD_COUNT, 93
relational operator, 14
REM, 22
REPLY, 82
RESTART, 45
RIGHT$, 31
ROM_SIZE, 96
RS-232 Communications, 76
RS-485 Communications, 80
Run-time Engine, 3, 5
Run-time Error, 122

S

SAVE_TRANSACTION, 86
SAVE_TRANSACTION_EX, 87
Scan Mode, 125
  Aiming Mode, 126
  Alternate Mode, 126
  Auto Off Mode, 125
  Auto Power Off Mode, 125
  Continuous Mode, 125
  Laser Mode, 126
  Momentary Mode, 126
  Repeat Mode, 126
  Test Mode, 126
SELECT_FONT, 73
SEND_WEDGE, 57
SET_COM, 78
SET_COM_TYPE, 78
SET_PRECISION, 22
SET_RTS, 79
SET_STATION_ID, 46
SET_VIDEO_MODE, 73
SET_WEDGE, 57
SGN, 23
SHIFT_LOCK, 69
SHOW_CALENDAR, 61
SHOW_IMAGE, 73
SMC_FREE_MEMORY, 96
SMC_SIZE, 96
START DEBUG, 96

START_NETWORK, 83
Station ID, 80
STOP BEEP, 58
STOP DEBUG, 96
STOP_NETWORK, 83
STR$, 33
string, 12
STRING$, 34
subroutine, 18

T

TIME$, 62
Time-Out, 126
Timer, 60
TIMER, 62
Transaction Files, 10
TRANSACTION_COUNT, 87
TRANSACTION_COUNT_EX, 87
TRIM_LEFT$, 31
TRIM_RIGHT$, 31

U

UCASE$, 33
UNLOCK, 44
UPDATE_RECODE, 93
UPDATE_TRANSACTION, 88
UPDATE_TRANSACTION_EX, 88

V

VAL, 33, 34
variables, 13
VERSION, 46
VOL, 59

W

WAIT, 62
Wedge Setting, 53
  Alphabets Case, 54
  Capital Lock Auto-Detection, 54
  Capital Lock Status, 54
  Digit Transmission, 55
  Digits Position, 54
  Inter-Character Delay, 55
  KBD/Terminal Type, 53
  Shift/Capital Lock Keyboard, 55
WHILE… WNED, 28
WRITE_COM, 79, 80